EVALUATION OF HIDDEN MARKOV MODEL FOR MALWARE BEHAVIORAL CLASSIFICATION

By

Mohammad Imran

A research thesis submitted to the Department of Computer Science,
Capital University of Science & Technology, Islamabad
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE
CAPITAL UNIVERSITY OF SCIENCE & TECHNOLOGY
ISLAMABAD
October 2016

To my parents and family

Contents

List of Figures

List of Tables

Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
API	**A**pplication **P**rogramming **I**nterface
DAML	**D**ARPA **A**gent **M**arkup **L**anguage
DBSCAN	**D**ensity **B**ased **S**patial **C**lustering of **A**pplications with **N**oise
DLL	**D**ynamically **L**inked **L**ibrary
DT	**D**ecision **T**ree
FDA	**F**isher **D**iscriminant **A**nalysis
FPR	**F**alse **P**ositive **R**atio
FNR	**F**alse **N**egative **R**atio
GA	**G**enetic **A**lgorithm
HMM	**H**idden **M**arkov **M**odel
IBL	**I**nstance **B**ased **L**earning
k-**NN**	k-**N**earest **N**eighbor
LSH	**L**ocality **S**ensitive **H**ashing
MCM	**M**arkov **C**hain **M**odel
MSA	**M**aximum **S**equence **A**lignment
OIL	**O**ntology **I**nterchange **L**anguage
PCA	**P**rincipal **C**omponent **A**nalysis
PHMM	**P**rofile **H**idden **M**arkov **M**odel
RF	**R**andom **F**orest
SOM	**S**elf **O**rganizing **M**ap
SVM	**S**upport **V**ector **M**achine
SWRL	**S**emantic **W**eb **R**ule **L**anguage
TPR	**T**rue **P**ositive **R**atio

Publications in support of this dissertation

Journal articles

1. **Imran, M.**, Afzal, M. T., and Qadir, M. A. (2016). Malware classification using dynamic features and hidden Markov model. *Journal of Intelligent & Fuzzy Systems*, Vol. 31, No. 2, pp. 837-847. IOS Press.

2. **Imran, M.**, Afzal, M. T., and Qadir, M. A. (2016). A comparison of feature extraction techniques for malware classification. *Turkish Journal of Electrical Engineering & Computer Sciences.* TÜBİTAK. (Accepted for publication).

Conference papers

3. **Imran, M**. Afzal, M.T., and Qadir, M.A. (2015). Using hidden Markov model for dynamic malware analysis: First impressions. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2015 12th International Conference on*, August 2015, Zhangjiajie, China. Pages 816–821. IEEE.

4. **Imran, M.**, Afzal, M. T., and Qadir, M. A. (2015). Similarity-based malware classification using hidden Markov model. In *2015 Fourth International Conference on Cyber Security, Cyber Warfare, and Digital Forensics (CyberSec)*, October 2015, Jakarta, Indonesia. Pages 129–134. IEEE.

Acknowledgements

First of all praise be to Allah, The most gracious, Who blessed me with the opportunity, capabilitiy and resources to pursue the doctoral programme, and it's all due to His grace that I saw it through.

One of the most notable of Allah's blessings upon me was in the form of my supervisor. PhD supervisors need a special skill; they must have night-vision eyes so that when the student gets totally lost in the pitch dark ravines of the research landscape, they can find a way and guide the lost soul to the right track. My supervisor has not only got such vision, he's got additional telescopic lens built in. When I announced less than a year and a half ago that I was quitting because I could not see where I was going, he was able to see the path that I needed to traverse in order to reach the summit. I don't have words to thank you, Dr. Muhammad Tanvir Afzal, for the motivation, guidance, support and encouragement that you provided to me on constant basis.

I am grateful to Dr. Muhammad Abdul Qadir, Head of the Center for Distributed and Semantic Computing (CDSC), whose advocacy of the "doer approach" led me to some quick decisions that saved me a lot of precious time. Very special thanks to Dr. Abdul Shahid, my senior research fellow at CDSC, who had more confidence in me than I had in myself. He boosted my morale at every point I was feeling shaky, which was just about every other day. I am also thankful to other members of CDSC whose discussion and constructive criticism maintained an environment that was conducive for research.

I am indebted to all the internal, external and foreign examiners for taking time to read and evaluate my synopsis and thesis, for identifying the shortcomings, and for providing their learned opinion on how to improve upon these.

There are numerous other people at C.U.S.T. who helped me in pursuit of my PhD in one way or the other including the faculty members at Department of

Computer Science, managerial and support staff, and the librarian to mention a few. Thank you all.

I owe a lot to my close friends, Dr. Mureed Hussain, Muhammad Masood Tahir and Ikramullah, for being constant source of inspiration and motivation throughout this time. There are so many other well wishers including friends, colleagues and relations who remembered me in their prayers. Allah bless you all.

In the end I must mention the being who cares more about me than I do for myself, my mother. I always felt her prayers by my side in the time of despair and frustration, and this feeling gave me energy to put myself together. And last but not the least, I have profound gratitude for my wife who had to face double dose of educational stresses: from my daughters as well as from me. May Allah reward you for your sacrifice and selflessness.

Abstract

Malware is a growing threat to computer systems and networks around the world. Ever since the malware construction kits and metamorphic virus generators became easily available, creating and spreading obfuscated malware has become a simple matter. The cyber-security vendors receive thousands of new malware samples everyday for analysis. It has become a challenging task for the malware analysts to identify if a given malware sample is a variant of a known malware or belongs to a new breed altogether. Since making an accurate decision about the nature of an unknown malware sample is crucial for updating of signature databases and propagation of the update to their customers, therefore vendors of cyber-security products need accurate malware classification techniques for this purpose.

The research community has been active for providing a solution to the above problem, and a number of diverse avenues have been explored such as machine learning, graph theory, finite state machines, etc. Furthermore, many syntactic and semantic aspects of computer programs have been tried out in search of the best aspect that could be used to distinguish between harmful and harmless computer programs, and to differentiate malware belonging to different families. All the proposed approaches have merits and demerits of their own, and the search for a solution that maximizes the classification accuracy with minimal computational costs is continued.

This dissertation formulates malware classification as a sequence classification problem, and evaluates a widely used sequence classification tool, Hidden Markov Model (HMM), for the task of malware classification. HMM has been a method of choice for a broad range of sequential pattern matching applications such as speech analysis, behavior modeling and handwriting recognition to name a few. The dissertation first proposes and evaluates novel methods of malware classification by combining HMM and malware behavioral features, which are attributes frequently used to distinguish between normal and malicious programs and to differentiate among malware families. As an another major contribution, the dissertation fills a significant research gap by studying the role of an important HMM parameter, the number of hidden states, in malware classification applications. Based on observations from comprehensive experiments conducted on a large and diverse dataset consisting of malware behavioral reports, the dissertation concludes that although HMM shows encouraging results when used for malware classification tasks, its potential from a practical standpoint is fairly limited. The dissertation makes the third contribution by proposing to replace the HMM component of malware classification method with Markov Chain Model (MCM), and performing comparative evaluation between the two models. Results of the comparison prove that classification performance achieved by HMM can be attained much more efficiently by MCM, and therefore MCM should be preferred over HMM for malware classification applications.

Chapter 1

Introduction

The aim of this research is to enhance the current state-of-the-art in malware classification using Hidden Markov Model (HMM) with malware behavioral features, and evaluate the prospects of such a combination from effectiveness and efficiency perspectives. This dissertation elaborates various experiments performed for achieving the goal of this research, that is, to propose and validate a novel method of malware classification and to study it from different angles in order to ascertain its feasibility for realistic application.

This chapter paints the background for the presented research by providing a broad introduction to the field of malware analysis, detection and classification, followed by a brief overview of literature. Based on observations made from the literature, a problem statement is formulated and a statement of motivation for the research follows. Contributions made by the presented research to scientific knowledge are highlighted next. The chapter concludes with the methodology adopted for conducting this research.

1.1 What is malware?

The term malware was introduced by Yisrael Radai in 1990[1]. This term is a shortened combination of the words *malicious* and *software*, and is used to represent software that is intended to cause harm to the computer systems[2]. A broader, more generic definition is provided by Moser et al. who describe malware as the software that “deliberately fulfills the harmful intent of an attacker” (Moser et al., 2007). The latter definition captures all the aspects and dimensions of the harm or damage that can be caused to not only the computer systems and networks, but also to their users.

[1]`http://www.pcworld.com/article/147698/tech.html`
[2]`http://technet.microsoft.com/en-us/library/dd632948.aspx`

As defined above, the most notable property that distinguishes malware from other software is the intent of its developer; malware is developed to perform tasks on a computer system against the will of, and without the knowledge of, the system's user. Malware is a representative term spanning the words like virus, worm, spyware etc., which are classified according to the modus operandi of the ill-intending software, and are discussed in detail shortly. The malicious software enters a target system through various means, known as the attack vectors, which will be discussed later.

The first ever malware may have been developed just as an individual act of mischief[3] or to prove technical superiority over a competitor (Egele et al., 2012), but the criminal mind did not take long to anticipate its potential benefits. As the use of computers quickly grew in all kinds of fields serving a variety of purposes, malware writers explored ways of illegally accessing and manipulating the computer systems with bad intentions. Gains behind such activities could be financial (stealing money through counterfeit bank transactions), commercial (accessing competitors' business data to gain insight into their policies) or strategic (destroying a country's nuclear setup through sending destructive commands to the control system (Langner, 2011)), to name just a few. The Symantec 2016 Internet Security Threat Report[4] quotes BofA Merrill Lynch Global Research for the annual global losses caused by malware to be up to $575 billion. The same report estimates the number of new malware variants created in 2015 to be approximately 430 million, showing a 36 percent increase over the 2014 figures. This amounts to an average of more than one million malware samples per day. Malware has become a very serious problem indeed.

A continuous battle, therefore, has been going on between the malware developers who are constantly trying to attack the computer systems and networks for their ill purposes, and the computer security community who are striving to provide the users of computer systems with the ultimate security solution through research

[3]`http://download.bitdefender.com/resources/files/Main/file/Malware_History.pdf`
[4]`www.symantec.com/security-center/threat-report`

and development. A review of the techniques and methods addressed at solving the malware problem is given in Chapter 2.

1.2 Types of malware

A look at the literature related to the malware problem reveals the use of various terms like virus, worm, Trojan horse etc., by the research community. Although collectively represented by the word malware, these terms are used to represent software with different characteristics. A short description of the most common terms and their distinguishing features is give below.

1.2.1 Virus

A virus is a software that mimics the spreading mechanism of its biological counterpart to replicate itself. The initial infection attaches a copy of the virus to an executable application, called the host application. Whenever the host application is executed, the virus also gets executed. Upon execution, the virus creates a copy of itself and attaches it to another application, thus spreading the infection. The virus may perform other tasks in addition to the replication, such as destroying files on the infected system. The infection spreads to other computers through shared media such as network links or USB devices etc.

1.2.2 Worm

The worm spreads like a virus by making clones of itself, but does not need a host application for execution. It executes independently and replicates not only inside the infected system but also sends its copies to other computers through the network.

1.2.3 Trojan horse

Named after the famous wartime technique used by the Greeks to invade the city of Troy, this type of malware works by cloaking itself as a legitimate, useful application. Once the user installs a Trojan on the system, the malware may destroy system files, replicate itself or harm the system in any other way.

1.2.4 Rootkit

Rootkits achieve their malicious objectives by elevating their privilege levels within the system. This higher level makes them invisible to most security procedures and allows them to perform the harmful tasks without getting noticed. Using the elevated privileges, Rootkits can hide system resources from the Operating System itself. Rootkits are typically not self-replicating.

1.2.5 Spyware

As the name suggests, spyware performs spying on the attacked system by retrieving sensitive and private data from the system, and sending it to the spyware controller through the network. The nature of data collected from the system depends on the objectives of the spyware; username-password combinations for websites may be sought for accessing online bank accounts, web browsing history may be collected for commercial uses, and private data or emails may be retrieved for any malevolent purposes.

1.2.6 Adware

This kind of malware displays unwanted and unsolicited advertisements to the users. The advertisements are shown in a new browser window or a pop-up window which could be difficult to close. While these malware disturb and irritate the users, they also serve another purpose - market and promote products.

1.2.7 Bot

A Bot, also known as Internet Bot or Web Bot, is software that works in a fashion similar to the spyware in that it connects to a remote master over the network. The function of a Bot is not only spying, but also controlling the target system under the command of the master. A single master can connect to and control a large number of Bots spread on the network, forming a Botnet, to perform malicious activity on large scale.

1.3 Malware attack vectors

Attack vectors are the methods using which an attacker gains access to, and implants a malware into, the target system. Egele et al. (2012) have identified three main categories of attack vectors described below.

1.3.1 Use of vulnerabilities

Vulnerabilities are flaws or bugs in software that allow an attacker to invade the system. Software vendors keep on searching for vulnerabilities in their software (system software as well as applications), and so do hackers. If the former find a vulnerability first, then a patch is issued to fix the flaw. On the other hand if the hackers discover the flaw before the others, then they use it as an attack vector to dispatch their malware to the systems running the vulnerable software.

1.3.2 Drive-by downloads

Users may be tempted by spam emails to visit some specific websites which have a mechanism to execute or download malicious code on the users' system through a loophole in the web browser. The user is not aware of the code being downloaded or executed, and as a result the attacker is able to infect the system with their malware.

1.3.3 Social engineering

Social engineering refers to the process of gaining someone's confidence in order to use them for malevolent activities. For example, an attacker may use a false identity, such as that of a bank's representative, to request a user to validate his/her online accounts through an email. In a similar fashion, users may be provoked to visit certain websites which contain harmful content.

1.4 Combating malware: Malware analysis

The initial techniques to counter the threat posed by malware were based on simple strategies, such as looking for a specific code pattern in a suspicious file.

Another way to prevent malware from replicating itself on a system was to make the malware believe that the system is already infected, by placing certain files in the system which were typically created by the malware. With the passage of time the malware became more advanced, and the above mentioned simple methods were unable to cope with the challenge. Therefore, the computer security community has been analyzing the malware deeply from various angles, in order to provide countermeasures against the malware.

The primary objective of malware analysis is to identify a property, or a set of properties, that distinguish the malware from normal, legitimate programs. Using such properties, detection techniques can be developed for identifying potential threats on a computer. Malware analysis has been performed using three approaches, namely static, dynamic and hybrid, which are described below.

1.4.1 Static analysis

Static analysis techniques try to identify features pertaining to program code that can be employed to distinguish malware from normal programs. For example, a simple static analysis technique may use plain code bytes as feature. The aim of such a technique will be to find a representative pattern of code bytes specific to a given malware. The pattern, referred to in literature as *signature* (Christodorescu et al., 2005), is then searched for in a given file to determine if the file is malware or benign. Different features that have been used for static analysis are given below.

- File Information: Information present in the file headers of a portable executable (PE) binary, such as a list of the library functions used by the program, is investigated by Schultz et al. (2001).
- Code Bytes: Raw bytes read from the binary file have been used as features by many researchers. Nataraj et al. (2011) have displayed the bytes as pixels, while n-grams of bytes (Santos et al., 2009), and byte sequences (Schultz et al., 2001) have also been exploited.

- opcodes: Short for operation codes, these are hexadecimal representation of the operation being performed in an instruction. opcodes are not readily accessible in a program binary, rather a tool like IDAPro[5] is used to disassemble the binary in order to get the opcodes. Research activities have used individual opcodes (Austin et al., 2013) as well as their sequences (Attaluri et al., 2009) and sequence frequencies (Santos et al., 2013).

- Instructions: Machine instructions obtained from disassembled programs have been used in terms of their sequences, frequencies (Ye et al., 2010) and counts (Hu et al., 2009) for malware detection.

- Functions: Some researchers have highlighted the importance of functions as representative features for malware detection. Programs can have functions of their own or can call external library functions. Relationship between calling and called functions has been used as a feature in malware detection (Kinable and Kostakis, 2011; Agrawal et al., 2012). Another approach is to use the length of functions in terms of number of bytes as feature (Islam et al., 2013).

- Basic Blocks: A basic block is a section of code which does not have any jumps or subroutine calls. In other words, instructions in a basic block are always executed in a sequence. In a disassembled program, basic blocks can be roughly described as code sections between two jumps or subroutine calls.

- Printable Strings: Programs generally have hard-coded ASCII strings in their code. A typical use of such strings is to store messages for the user, etc. Islam et al. (2013) and Schultz et al. (2001) have explored the possibility of using strings found within the binaries for the purpose of malware detection.

- DLL Information: DLLs are dynamically linked libraries containing functions to be used by application programs. Schultz et al. (2001) have used the DLL related information present in headers of PE files as features. The features included the DLLs accessed by the malware sample being analyzed,

[5] `https://www.hex-rays.com/products/ida/`

the function calls made to DLLs, and the total number of functions exported by the DLLs.

The major benefit of static analysis is its effectiveness in detecting known malware, while the obvious drawback is its requirement for frequent signature updates, and hence the inadequacy for detecting zero-day threats. Static analysis is particularly vulnerable to code obfuscation techniques (Moser et al., 2007). To deceive the malware detectors based on static features such as bytes or opcodes, the malware developers use techniques to change the code of the program binary without changing its function. Another objective of code obfuscation is to make the task of reverse engineering a program binary difficult.

One of the obfuscation techniques is code packing (You and Yim, 2010) which refers to the process of compressing or encrypting the harmful code as data, and attaching with it an unpacking or decrypting engine. When the program executes, the unpacking engine decompresses the compressed malware code and executes it. This kind of packing is detectable because although the encryption of the malware code is made to look different by using different encryption keys, the decrypting engine usually stays the same.

In order to counter the detection of common decryptor, it is mutated between generations of the same malware. This polymorphic decrypting engine performs the same action as before, but looks different because of techniques such as dead code insertion, register renaming, subroutine re-ordering, instruction substitution, etc. Such malware can be detected by analyzing its execution; after decrypting the malicious code, the executing code can be traced in memory.

A more advanced obfuscation technique, termed as metamorphic transformation, mutates the whole program instead of just the decrypting part, using code obfuscation methods of dead code insertion etc. This type of malware is most difficult to detect because it looks different not only when encrypted, but also when it is decrypted and is being executed. Detection of metamorphic malware is an active area of research in cyber security domain (Agrawal et al., 2012; Wong and Stamp, 2006). In summary, use of static analysis alone may not be enough to counter the malware threat, as maintained by Moser et al. (Moser et al., 2007), who advocate the need for dynamic analysis.

1.4.2 Dynamic analysis

The term dynamic analysis represents all those techniques which analyze a program by executing it. Another term which commonly replaces the dynamic analysis is behavior analysis, truly representing the objective of this kind of analysis which tries to capture the behavior of a program. In contrast with static analysis approaches which use the syntactic representation of files, dynamic analysis focuses on semantics of a program. Another way of expressing the difference is that the static analysis attempts to find what malware *looks like*, while the dynamic analysis tries to discover what the malware *does*.

In order to create a behavioral profile or representation, the malware is executed in a sandbox, which is an environment that allows controlled execution of programs by limiting access to system resources. A virtual machine is an example of a sandbox, in addition to many other tools specifically built for analysis of malware behavior such as CWSandbox (Willems et al., 2007), ANUBIS[6], etc.

Some common features explored by dynamic analysis methods are given below.

- Instructions: While machine instructions are also observed in static analysis, their use in dynamic analysis is slightly different. Here the instruction traces executed during a sample's execution are recorded in order to use them as features (Anderson et al., 2012; Rieck et al., 2011).

- System Calls: In order to perform a privileged, system level operation, an application makes a request to the operating system through a system call. System calls have been actively explored as an effective means of identifying a program's behavior. Different techniques have been proposed which use

[6]`http://anubis.seclab.tuwien.ac.at`

system calls aimed at different categories of operations. Some examples are given below.

  - File operations (Chandramohan et al., 2012; Saxe et al., 2012; Huang et al., 2014, 2011; Stopel et al., 2006)
  - Registry operations (Chandramohan et al., 2012; Saxe et al., 2012; Huang et al., 2010, 2011)
  - Process operations (Chandramohan et al., 2012)
  - Network operations (Bayer et al., 2009; Chandramohan et al., 2012; Saxe et al., 2012; Huang et al., 2010, 2011; Stopel et al., 2006)
  - Processor features (Stopel et al., 2006)

- API Calls: Application Programming Interface (API) is a mechanism which offers a simple interface to applications developers for accessing system resources. For example, the network API includes functions to open and close sockets, and to send and receive data using opened sockets. Like system calls, API calls made by a program can also be used to observe a program's behavior. Research efforts depending on API calls for analyzing malware behavior include (Islam et al., 2013; Rieck et al., 2008; Trinius et al., 2009).

- Strings in Memory: Use of ASCII strings in static analysis is limited to those strings that are found in program binaries. In case of compressed or encrypted code, static analysis will not be able to reveal any such strings. However, malware may reveal some strings during execution which can be detected by inspecting program memory. This hypothesis has been tested in analysis methods proposed in (Karampatziakis et al., 2013) and (Dahl et al., 2013).

- Memory Writes: The partial and full results of calculations and computations, performed by a program during execution, also have the potential of depicting a program's behavior, as demonstrated in (Grégio et al., 2013).

One obvious edge that dynamic analysis has over the static approach is that it can handle the case of obfuscated malware, since the obfuscation only results in code that looks different; the functioning of code does not change. For this reason, dynamic analysis is believed to be able to capture the higher level program behavior (Rieck et al., 2011), and therefore is much more effective for malware detection and classification purposes.

Although the dynamic approach overcomes the limitations of static analysis, it has its own share of shortcomings. A major issue is that each malware sample must be executed long enough so that a substantial number of activities can be recorded, enabling a reasonable representation of the sample's behavior. But there is a limit to how long each sample can be run because there may be thousands of samples to be analyzed, and a short execution may not allow complete behavioral capture. Even if a sample is executed under no such time constraints, there is no guarantee that the malware will reveal its true behavior. The malware may be dependent on some particular event for its malicious activity, or may become active on a specific moment in time, or may halt its operation until commanded from a remote system. In other words, the malware may not follow its complete execution path, therefore rendering the analysis incomplete (Egele et al., 2012).

To make matters more complicated, the malware developers make the code sense the execution environment; if the execution is being done within a virtual machine or with an automated analysis tool, the malware may behave innocently (Egele et al., 2012).

Yet another approach taken by malware developers for evading dynamic analysis is to use stalling code (Kolbitsch et al., 2011). This strategy exploits the limitation of time for which the malware sample is executed. The purpose of stalling code is to delay the execution of malicious portion of the malware code long enough so that the behavioral monitoring system is not able to record anything of significance during the (typically) short analysis.

1.4.3 Hybrid analysis

Keeping in view the insufficiency of static and dynamic analysis approaches, hybrid techniques have been proposed which make use of both types of analyses. In hybrid approach, first the static features are identified and then the malware is executed for capturing its dynamic behavior. Hybrid techniques can be applied in different ways. Static and dynamic feature vectors can be stored and analyzed separately, and then the results may be aggregated (Anderson et al., 2012; Hu and Shin, 2013). Another approach is to combine the static and dynamic features into one vector to perform a composite analysis (Islam et al., 2013).

1.5 Combating malware: Malware detection and classification

Identification of important static or dynamic features pertaining to programs is just the first step towards the solving the malware problem. In the next step, the identified feature (or the set of features) is used in some process for separating malware from benign programs. Such processes are generally considered to be performing malware *detection*. With the number of new malware variants being reported everyday reaching close to a million, the task of malware *classification* has become a challenge for the anti-malware industry. An evidence of this fact is the Microsoft malware classification challenge 2015[7], inviting the researchers to brainstorm and suggest the best possible mechanism of classifying among malware families.

A vast number of new malware are obfuscated variants of existing malware (Elhadi et al., 2014), and show similar behavior traits despite having different code. Such variants developed from the same base are said to belong to the same malware *family*. The developers of anti-malware products, thus, need to identify families of malware so that signatures can be efficiently generated on family basis. Another important benefit of classifying malware is that trend of malware spread can be anticipated and corresponding measures may be taken in advance.

[7]`https://www.kaggle.com/c/malware-classification`

Research efforts have been made in various dimensions for performing malware detection and classification. Although Chapter 2 discusses in detail these research activities, a brief account of the reviewed literature is provided here to paint an overall picture. Machine learning has been quite effectively employed for performing malware classification. Researchers have used both the supervised and unsupervised types of machine learning techniques in order to identify and classify malware. Among the supervised category, classification (Schultz et al., 2001; Rieck et al., 2008; Santos et al., 2013; Islam et al., 2013; Karampatziakis et al., 2013), Artificial Neural Networks (Stopel et al., 2006; Dahl et al., 2013), and Hidden Markov Models (Wong and Stamp, 2006; Austin et al., 2013; Annachhatre et al., 2014) are dominant, whereas the unsupervised learning method of clustering (Bayer et al., 2009; Ye et al., 2010; Rieck et al., 2011; Hu and Shin, 2013) has also been used with good results.

Some researchers have taken the graph theoretic approach to detect malware (Hu et al., 2009; Agrawal et al., 2012; Park et al., 2013). Genetic Algorithms are an efficient tool for optimization problems, and have been applied to the malware classification domain as well (Mehdi et al., 2009; Kim and Moon, 2010).

The disciplines of information visualization and ontologies may seem to be unrelated to the problem of malware analysis and detection, yet the literature includes research efforts being carried out in this regard. Visualization-based methods are suggested by (Yoo, 2004; Trinius et al., 2009; Nataraj et al., 2011; Saxe et al., 2012) while the use of ontologies is demonstrated in (Huang et al., 2011, 2014).

1.6 Observations from the literature

Critical observations from the literature survey can be summarized as:

1. Despite high accuracy figures being reported by the current state-of-the-art in malware research, malware is spreading with a threatening pace and consequently hundreds of thousands of new malware are reported everyday. It

poses research challenges to comprehensively investigate the issues in malware classification.

2. Literature suggests that malware classification techniques based on dynamic analysis may perform better than those based on static approach for classification of unknown malware (Bayer et al., 2010; Egele et al., 2012; Damodaran et al., 2015). Furthermore, the schemes which use machine learning are generally more accurate on detecting and classifying unknown malware. Based on these two observations, it seems likely that a malware classification scheme based on a combination of machine learning and behavioral analysis approaches may be able to enhance the current state-of-the-art.

3. Since the behavior of a computer program can be expressed as a sequence (of instructions, system calls, etc.), therefore malware classification can be represented as a sequence classification problem. For classifying sequences, an effective and widely used machine learning technique is Hidden Markov Model (HMM) which has proven its capabilities in application areas such as speech recognition, human behavior modeling, and protein sequencing etc.

4. While the above observations suggest that using HMM with sequential representation of malware's dynamic behavior could prove to be effective for classification of malware, only a few research efforts have been reported in the literature which make use of such a combination. Most of the previously proposed malware classification schemes based on HMM rely on static malware features.

5. The previously proposed HMM based malware classification schemes are generally evaluated on small datasets.

6. No analysis is performed on the role of hidden HMM states for malware classification applications, which is a key parameter in HMM modeling.

7. Previously proposed malware detection and classification techniques based on HMM only focus on accuracy, and the computational costs are usually ignored.

1.7 Problem statement

The observations from the literature noted above can be summarized in the form of a problem statement as:

- To evaluate Hidden Markov Model for the task of malware classification using malware's behavioral features from effectiveness, analytical and efficiency aspects.

The following research questions have been identified and addressed in this dissertation in order to accomplish the task described in the problem statement.

1. How can HMM be used to classify malware on the basis of their behavior?
2. What is the role of number of hidden states when using HMM for malware classification?
3. How efficient is HMM based malware classification to be used in practical situations?

The first research question is discussed in Chapter 3 of this dissertation which proposes and evaluates an HMM-based sequence classification method for classification of malware. Chapter 3 performs a comparative analysis of the proposed classification method with other methods including the state-of-the-art. Chapter 4 extends the evaluation by comparing the similarity-based classification method with another sequence classification technique commonly used in the literature, the feature-based classification. Chapter 5 provides details of experiments carried out in an attempt to answer the second and third research questions.

1.8 Motivation

The motivations behind the proposed research are manifold. We first discuss the motivating factors behind conducting research in malware analysis and classification in general, and then elaborate our reasons for investigating Hidden Markov Model within this domain.

1.8.1 Why malware analysis?

A security report issued by Symantec Corporation states the number of new malware threats found in 2015 to be around 430 million, at the alarming rate of more than one million per day. This fact illustrates that the currently available solutions are not able to stop the malware tsunami. That is why the research community is striving to find an ultimate solution to the malware problem. This research is an attempt to contribute in the search for the perfect anti-malware solution.

The worldwide cyber-security industry is reported to be worth 122 billion US Dollars in 2015[8]. These figures are expected to escalate even more due to the rise in malware threats and the global losses caused by the malware. It can therefore be inferred that there will be a worldwide demand of cyber-security professionals and experts in the future. There is a need to promote the understanding of and research on malware so that expertise is developed in order to meet the demand.

1.8.2 Why Hidden Markov Model?

Among the scientific disciplines explored for malware detection and classification tasks, machine learning has been of particular interest among the researchers. The reason for this focus on machine learning is that the majority of new malware is variant of existing malware, and similarity in malware behaviors can be effectively utilized to detect even previously unknown malware which is a key requirement in today's world where millions of new malware are reported every day. This is a prime advantage of machine learning based methods over conventional methods of malware detection and classification, which are only able to detect known malware.

As discussed earlier, previous research suggests that dynamic malware features may be able to represent a malware's behavior more accurately than static features, and therefore can be used more effectively to differentiate among different malware families. Some dynamic features, such as system calls or API calls generated by a malware while execution, represent the malware's behavior in the form of a sequence. This sequential nature of dynamic features motivates us to treat the

[8]`http://www.marketsandmarkets.com/PressReleases/cyber-security.asp`

malware classification problem as a sequential classification problem. Classification of sequences is a well-studied and well-researched problem in many sequential pattern matching application areas such as speech processing, handwriting recognition and behavior modeling, etc., and a variety of solutions based on machine learning have been proposed in the literature to address this problem. *This research investigates if and how a machine learning based solution from sequence classification paradigm can be effectively and efficiently used to classify malware.*

Xing et al. (2010) have classified sequence classification methods into three broad categories, namely feature based, distance based and model based classification. The feature based classification involves selecting some features from the given sequences in the form of a fixed length feature vector and then applying conventional classification methods on the feature vectors. Feature vectors formed by counting presence or frequencies of unique elements in the sequence, or of short subsequences (k-grams) are most commonly used in such methods. In the distance based classification methods, a distance metric is computed to measure similarity between two given sequences and this distance metric is then used in classifiers like Support Vector Machine (SVM) and k-NN for classification of sequences. Euclidean distance and Dynamic Time Warping distance are two examples of such metrics. The third category of sequence classification methods is the model based classification, in which the sequences are assumed to be generated by a probabilistic model. Such a model is first estimated from given sequences and then probability of a given sequence being generated by a given model determines the classification decision. Hidden Markov Model is a prominent example of such classification methods.

While the success and effectiveness of feature based and distance based methods relies on selection of suitable features and distance metric respectively, the model based approach treats the sequence as a whole and does not depend on any modified or reduced representation thereof. Furthermore, a survey of previous research in aforementioned sequential pattern matching applications reveals that Hidden Markov Model is one of the most widely used techniques for such applications.

Inspired by the above observations and research gaps identified through a comprehensive survey of literature, this dissertation focuses on use of HMM for malware behavioral classification, and tries to investigate its feasibility from various angles.

1.9 Contribution of research

This dissertation makes the following contributions to scientific knowledge:

- The dissertation proposes and evaluates a novel method of malware classification by combining HMM and malware behavioral features using real malware data.

- The dissertation fills a significant research gap by studying the role of an important HMM parameter, the number of hidden states, in malware classification applications. Based on observations from comprehensive experiments conducted on a large and diverse dataset consisting of malware behavioral reports, the dissertation forms an opinion that although HMM shows encouraging results when used for malware classification tasks, its potential from a practical standpoint is fairly limited.

- The dissertation proposes an alternative, yet novel, malware classification method based on Markov Chain Model (MCM), and performs comparative evaluation between HMM based and MCM based methods from effectiveness and efficiency aspects. The dissertation concludes that MCM is a better method to model malware behavioral than HMM from efficiency perspective, and therefore former is more practical for malware classification applications.

1.10 Research Methodology

From the application point of view, the research presented in this thesis can be termed as applied research: Solution to an immediate problem is proposed and evaluated. From the objectives perspective, this research belongs to the exploratory category, since the objective of this research is to explore the *why* (or why *not*) of using HMMs for malware classification. In order to find answers to research questions, the methods used in this research are mostly quantitative; experiments are performed and results are analyzed to derive conclusions and form opinions. Qualitative analysis is also performed in order to find the rationale behind specifying the number of hidden states when modeling hidden Markov models for the malware classification task.

In conducting this research, the three-phase, eight-step model has been followed as proposed by Kumar (2011) with slight modifications as per the requirements of this research. The activities carried out during the course of this research are described below, and a mapping between these activities and Kumar's model is also highlighted in Figure 1.1

Phase I - Deciding what to research

Step 1: Formulating a research problem: This step consisted of three tasks:

1. Literature review
2. Definition of criteria for evaluation of HMM based malware classification methods
3. Identification of research gap

Step 2: Writing a research proposal: In the original model proposed by Kumar, this step is listed as *Step 5* in **Phase-II**. During the course of this research, however, the proposal for devising and evaluating a novel HMM-based malware classification method was formulated as the next step after identifying the research problem.

Phase II - Planing the research study

Step 3: Conceptualizing a research design: The research design for conducting this research was formulated in the form of the three research questions discussed

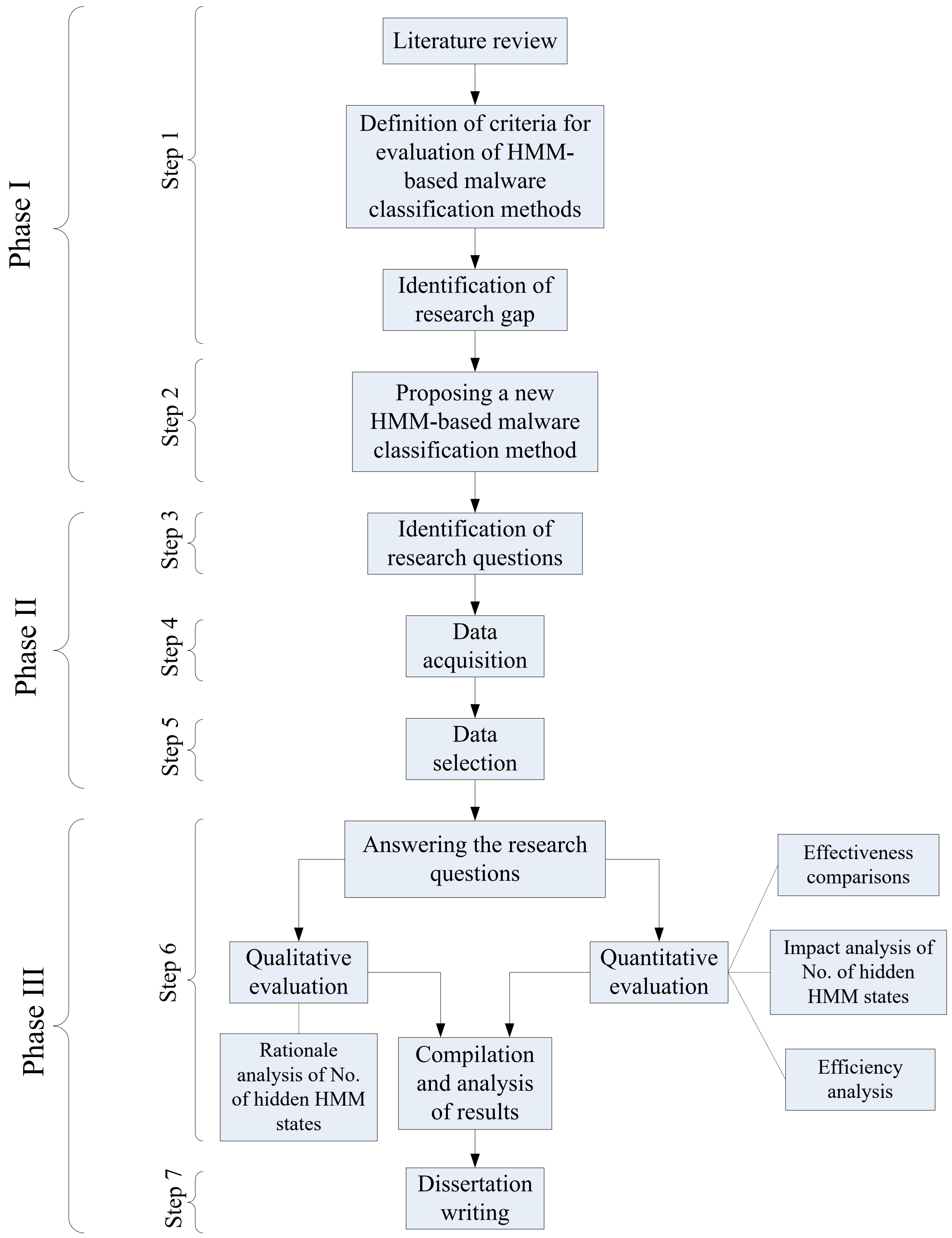

Phase I
Step 1
Literature review
Definition of criteria for evaluation of HMM-based malware classification methods
Identification of research gap
Step 2
Proposing a new HMM-based malware classification method
Phase II
Step 3
Identification of research questions
Step 4
Data acquisition
Step 5
Data selection
Phase III
Step 6
Answering the research questions
Qualitative evaluation
Quantitative evaluation
Effectiveness comparisons
Impact analysis of No. of hidden HMM states
Efficiency analysis
Rationale analysis of No. of hidden HMM states
Compilation and analysis of results
Step 7
Dissertation writing

Figure 1.1: The methodological steps for the proposed research

earlier. The methodology for addressing the questions is discussed in the corresponding Chapters. Keeping in view the research questions, the study design can be termed as quantitative; for research question 2, however, the study takes a qualitative approach.

Step 4: Constructing an instrument for data collection: Instead of creating a new dataset, a freely available large and comprehensive dataset was acquired which has also been used in previous research. For this reason, *Step 6* (data collection) in **Phase-III** of the original model has been skipped.

Step 5: Selecting a sample: Some of the malware families in the acquired dataset had very few samples. On the other hand, a few malware families had considerably large number of samples. Therefore a careful selection of malware families was performed in order to have a balanced dataset for conducting the experiments. This process is described in detail in Chapter 3.

PHASE III - Conducing the research study *Step 6: Processing and displaying data*: This step encompasses all the experiments performed for evaluating the proposed method against other methods (Chapter 3), for studying the impact of hidden HMM states (Chapter 4), and for analyzing the efficiency of the method (Chapter 5).

Step 7: Writing a research report: This dissertation is the output of this last step, in which activities of all the previous steps are described and their outcome is analyzed.

1.11 Dissertation organization

This dissertation is structured as follows. Chapter 2 sheds light on research efforts carried out in the domain of malware analysis and critically reviews the classic as well as state-of-the-art techniques proposed in the literature. Chapter 3 focuses on Hidden Markov Model and proposes a method of malware classification based on HMMs. Chapter 4 treats HMM as a feature extraction method and reports the results of experiments for comparing HMM against various commonly used methods of extracting features from sequences. Having ascertained the effectiveness of HMM for malware classification, results of a study on the effect of hidden HMM states on malware classification performance are reported in Chapter 5 which also discusses the time requirements for HMM modeling and testing, especially with respect to the number of hidden HMM states. Based on results drawn from the study, the dissertation proposes to use Markov Chain Model (MCM) in place of Hidden Markov Model and compares the two approaches with an aim to determine if MCM can be a better alternative than HMM for malware behavioral classification. Chapter 6 summarizes and concludes the dissertation.

Chapter 2

Literature Review

In this chapter various approaches for performing malware analysis and detection proposed in the literature are discussed. The main objective of the literature survey presented here is to get a broad overview of the efforts being made in this domain. The proposed techniques have been classified into different categories according to the particular area of scientific research from which these techniques have been inspired.

2.1 Clustering

Clustering is an unsupervised machine learning technique used to identify structure, or classes, within data (Han et al., 2006). As the name suggests, the technique aims at dividing the input data into clusters in such a way that similar data items are assigned to the same cluster. The main use of clustering in malware analysis and detection domain is to group known malware samples into classes, usually called families. Representative samples from each cluster, or family, are then analyzed to create signatures for detection of unknown malware.

A scalable, clustering-based malware behavior analysis system has been proposed by Bayer et al. (2009). They create behavioral profiles for known malware using ANUBIS. These profiles are generated on the basis of invoked system calls, their inter-dependencies and network activities, and represent behavior in terms of OS objects and operations. These profiles are then input to the single-linkage hierarchical clustering algorithm which groups malware binaries according to the exhibited behavior. To make the clustering scalable, the authors have used the Locality Sensitive Hashing which provides an approximate but efficient solution. The proposed system was able to cluster around 75,000 malware samples in 2 hours and a quarter. The accuracy of the proposed clustering method has been expressed in terms of quality which is defined to be a product of precision and recall, and is reported to be 0.959. Despite showing good results in evaluation, the proposed technique may be vulnerable to common evasion methods adopted by malware developers against behavioral analysis systems, as previously discussed.

Ye et al. used an "ensemble" of different clustering algorithms for achieving a better clustering of malware samples (Ye et al., 2010). They opted for static analysis of malware binaries, and selected instruction frequencies and instruction sequences within individual functions as the features for clustering. The proposed system combines hierarchical and k-medoids clustering algorithms, and uses a weighted-subspace k-medoids clustering on instruction frequencies and instruction sequences respectively. A target, consensus partition is obtained by combining the generated clusterings, termed as partitions. The precision values of all the clusterings used in the ensemble are averaged to get precision of final clustering, and is reported to be 0.9369. The results are achieved by incorporating sample-level constraints which require involvement of human experts, and this may limit the scalability of the proposed solution.

Rieck et al. (2011) combine supervised and unsupervised machine learning techniques for malware analysis. The dataset consists of malware behavioral reports, generated through executing malware binaries in CWSandbox (Willems et al., 2007) and converted to MIST (Trinius et al., 2009) format. The authors have proposed and tested a scheme which employs an approximation strategy using prototypes. After prototypes have been extracted from the behavioral reports, the next step assembles the prototypes into clusters using complete linkage hierarchical clustering, and then classifies the unknown binaries according to the identified clusters using what they term as the nearest prototype classifier. After obtaining an initial clustering, the classification and clustering steps are performed in an interleaved manner in order to perform incremental analysis. The authors maintain that the approximation approach makes their technique scalable without compromising accuracy. They tested the proposed solution on a reference dataset containing around 3000 malware binaries and reported an F-measure of 0.95 for clustering, and 0.981 and 0.997 for classification of known and unknown malware classes respectively. It is interesting to note that the results of their method for the larger dataset of more than 30,000 malware samples have not been reported. Since Rieck et al'.s method makes use of both the clustering and classification paradigms, therefore it has been included twice in the literature review summary provided in Table 2.1.

Chandramohan et al. address the scalability problem in the context of behavior-based malware clustering (Chandramohan et al., 2012). Such clustering techniques have to cater for an enormous feature space, thanks to the tens of thousands of new malware variants being created every day. The authors suggest limiting the feature vector size by first focusing on four types of system resources, and then abstracting individual operations on these resources into higher level operation types. In this way, the feature space becomes independent of the number of malware samples. The extracted features are represented as bit vectors. To achieve scalability from another perspective, the authors have used prototype-based clustering as proposed by Rieck et al. (2011). The behavioral reports used by Chandramohan et al. are generated through ANUBIS. The proposed system's performance in terms of F-measure has been reported to be 0.948 against a reference labeling of Ikarus[1] anti-virus. The method of feature encoding employed by the proposed scheme converts the given system call sequence into a fixed width feature vector containing binary values, representing the presence or absence of individual system calls in the sequence. This scheme loses the sequence information from the behavior reports which may render the dynamic analysis ineffective for some cases.

Hu et al. (2013) have proposed a hybrid approach using both static and dynamic features for performing malware clustering. The frequencies of opcode n-grams extracted from the disassembled and unpacked binaries are used as static features, and clustering is performed on the basis of these frequencies. The prototype-based clustering method is adopted from Rieck et al. (2011). For obtaining behavioral features, the malware is executed inside a VMWare[2] virtual machine and system

[1]`http://www.ikarussecurity.com/`
[2]`http://www.vmware.com`

calls are recorded. The n-grams of these system calls are then represented as fixed-length bit vectors for performing prototype-based clustering. The two clusterings are then combined into an ensemble using four different methods, and the results are compared against individual clustering schemes. The authors have reported a marked increase in malware coverage although the precision of the ensemble approach remains comparable to individual clustering algorithms.

Another attempt at employing clustering for malware classification has been presented in (Qiao et al., 2014). The proposed system, called CBM, transforms the dynamic traces composed of API calls into a particular byte-based sequential format (termed as BBIS) and then makes use of modified Malheur (Rieck et al., 2011) to perform clustering. The modifications in Malheur were required in order to make it work for data represented in BBIS format. The authors conclude that with the suggested modifications, the system was able to achieve a F-measure of 90.9%. Although this figure is less than that reported by Malheur, the authors maintain that it offers a strong and open source alternative to the current state-of-the-art.

2.2 Classification

Classification is a supervised machine learning technique which is used to predict a class label for an unlabeled data item (Han et al., 2006). This is a two-step process: In the training or learning phase, the system is provided with class labels and data pertaining to respective classes. The system learns and identifies the features or patterns distinguishing the individual classes. In the testing phase, an unknown sample is input to the system which tries to classify the sample on the basis of the learned patterns.

The malware analysis and detection problem lends itself quite naturally to application of classification (Willems et al., 2007). In case of binary classification, it needs to be decided if a given file falls into the malware class or the benign one. For the scenario of multi-class classification, the task is to identify which family of malware the unknown sample belongs to. The first step in classification is to identify some features which best identify a malicious file from a benign one. A large feature set can affect the performance of the classifier, therefore feature selection and feature extraction methods are sometimes used prior to training the classifier.

In one of the pioneering works on application of data mining techniques in malware analysis and detection, Schultz et al. (2001) describe the efficiency of classification technique in detection of unseen malware. They tried a number of static attributes of files such as parameters related to DLL usage, strings embedded within binaries, and binary code sequences, as features using multiple classifiers including RIPPER, Naïve Bayes and Multi-Naïve Bayes on a dataset containing 4266 files. According to their experiment, use of byte sequences as features in Multi-Naïve Bayes classifier achieved a detection rate of 97.76% with a false positive rate of 6.0% and an overall 96.8% accuracy. compared with signature-based detection rates of less than 50% on the same dataset, this result was a clear indication that data mining and machine learning techniques had the capability to be used for malware analysis and detection, and hence opened the avenues for further research in this field. However, due to use of static features, this technique is susceptible to obfuscation techniques.

Rieck et al. (2008) use a supervised machine learning technique, the Support Vector Machine (SVM), to determine the class or family of a given malware. As the first step of the process, malware samples are "attracted" and collected using honeypots, and then are labeled using an anti-virus software which assigns a malware class to each sample. Next, runtime behavior of the collected malware, in terms of API calls, is recorded by executing it in a sandbox and features are extracted from the stored behavior profiles. Features are embedded into a high-dimensional vector space and SVMs are trained for each malware family using the assigned labels. The results of individual classifiers are combined to obtain an overall classification. The scheme was tested on a corpora of 10,000 samples and was found to be 88% accurate. The scheme showed a relatively low accuracy, in addition to the obvious dependence on behavioral model which may not always represent the full malware behavior.

Santos et al. proposed to use opcode sequence frequencies, extracted through static analysis of portable executables (PE), as the representative features in supervised machine learning for the purpose of malware detection (Santos et al., 2013). The multi-step process starts with counting the opcodes found in disassembled PE files contained in separate malware and benign sets. Each opcode is then assigned a relevance value against both the malware and benign classes using the idea of mutual information. Next, opcode sequences frequencies are computed for opcode sequences from each file and are weighted by the relevance value of each opcode in the sequence. This yields a vector containing pairs of opcode sequences and the weighted opcode sequence frequencies, which, in turn, is fed as input to various classifiers. The authors compiled the results of using different classifiers such as Decision Trees, SVM, k-NN, etc. in WEKA (Garner et al., 1995), and reported an accuracy of more than 95% in case of SVM with sequence length (n) of 2. As the number of features increased exponentially with increasing n, experiments were not attempted for larger values for n. Better results could be achieved for larger values of n by applying feature reduction and feature selection steps on the feature set.

Researchers proposing solutions for malware analysis problem have focused on either static or dynamic feature extraction from malware samples for representing malware behavior. Islam et al. (2013) have used a combination of both types of features. The static features include the lengths of functions, in bytes, and the count of various printable strings found within the binaries. The only dynamic feature used is the frequency of API calls, made by the program during execution, against a global list of API calls made by all the samples. The vectors corresponding to the three features are then concatenated in order to form a combined vector which is then used in four different classifiers including Support Vector Machine (SVM), Decision Trees (DT), instance-based (IB1), and Random Forest (RF). Evaluation of the proposed system on two different datasets (old and new) proved that the combined static and dynamic features better represent the characteristics of malware binaries, especially when used with RF classifier which produced an accuracy of 97.055%. In comparison, use of individual features could reach at most 90%, again in case of using RF classifier on dynamic feature. However, the accuracy of the system was better on the old dataset than for the newer one, indicating its inability to effectively detect evolved malware.

Karampatziakis et al. employ file relationships for identifying unknown malware (Karampatziakis et al., 2013). They specifically target file archives and proceed in three steps. First a baseline logistic regression classifier is trained on labeled malware and benign files, and individual unknown files are assigned probabilities of being malware or otherwise. Features used by this classifier are extracted through dynamic analysis of executables, and include system calls along with parameters passed, tri-grams of system call sequences, null-terminated patterns in process' memory, and a bit vector representing some trivial metadata. Next, the archives containing the individual files are assigned probabilities through a container classifier. Finally, a relationship classifier improves the classification accuracy of each file using the file's previously computed baseline probability, and the container probabilities of all the archives which this file belongs to. The proposed system performed the classification on a dataset of 3.4 million files in a mere 16 minutes. Using the relationship-based classifier, the FNR was 15.2 % as compared to 42.1% achieved by the baseline classifier, while keeping the FPR to 0.2% in both cases. The proposed technique is based on the assumption that the probability of a benign file being classified as malware would be higher if it is contained in an archive which has one or more malicious files. This assumption may result in a high FP rate, depending on the ratio of malign and benign files in a given archive.

2.3 Graph theory

Graph theory has been applied to a diverse range of areas, including computer networks, route planning, electronic circuit design, to name just a few. Any problem that can be modeled in terms of data items (vertices) inter-connected through some kind of links (edges) can be converted into a graph problem, and therefore can be solved by applying principles from the graph theory. The following techniques for malware detection represent programs in the form of graphs, and then make use of graph theory to determine if a given program is malware or benign.

Hu et al. use call graphs to perform static malware detection at a higher-level of abstraction, instead of byte or instruction-level matching (Hu et al., 2009). By identifying function boundaries in disassembled program binaries, each program is represented as a graph depicting the call and callee relationships between functions, so that the functions are expressed as vertices and a directed edge goes from the called function to the callee. A given program's call graph is compared with those of known malware to determine its nature through performing nearest-neighbor search. The authors introduce an approximate edit-distance similarity metric, based on optimized Hungarian algorithm, to perform efficient comparison between graphs. Searching is made scalable by introducing a hierarchical indexing of graphs. At the higher level, graphs are indexed into a B+ Tree where each leaf node groups similar graphs according to code attributes like the instruction count and function count, etc. The second level indexes graphs within a leaf node of the B+ Tree in the form of a Vantage Point Tree, using the approximate edit distance metric. This combination of multi-level indexing and approximate distance metric allows for pruning of search space at higher level on the basis of easily computed attributes, and an efficient search for nearest neighbors within the graph database. Performance of the proposed indexing scheme has been reported using various parameters; the average response time for a query on the database of 100000 graphs was 21 seconds, and more than 90% of the queries successfully identify the query graph as representative of a malware. Being a static analysis approach, it may fail against sophisticated obfuscation methods, as well as novel code packing techniques, being used by malware developers. Also, since the technique depends on neighboring nodes to make a decision about a particular file's nature, it may lead to high FP rate.

Park et al. (2010) proposed a graph-theoretical solution to the problem of behavior-based malware detection. The technique involves creating a behavioral profile for a program on the basis of system calls and their arguments, and representing the profile in the form of a directed graph. The nodes of the graph depict system calls, and the edges signify dependence between system calls. Two system calls are considered to be interdependent if the output parameter of one is the input argument for the other. The authors used a metric based on the maximal common subgraph distance for representing dissimilarity between two graphs. This metric is then employed by the proposed classification algorithm that tries to group the graphs into malware families. Evaluation of the proposed solution on six families of malware produced promising results for most families; the less satisfactory classification in some malware families is attributed to the inherent behavior specific to those families, as identified by earlier research (Kolbitsch et al., 2009). The dataset used for the experiment is rather small (only 300 files belonging to 6 families) and a more exhaustive evaluation is therefore needed.

Another attempt to use graph theory for performing malware detection was made by Kinable et al. (2011). Similar to the method proposed by Hu et al. (2009), Kinable et al. used the function calls and their inter-relationship within a program binary to populate a graph representing the behavior of the program. A graph edit distance metric was introduced which computes the similarity between two given graphs on the basis of certain cost functions. This metric was used, with an approximation technique based on Simulated Annealing method of graph matching, as the distance function for application of two clustering algorithms, k-medoids and DBSCAN, on the graph dataset to identify malware families. On the basis of these experiments, the authors opined that k-means clustering is not suitable for malware analysis because of the need to find an optimal k. The performance of DBSCAN clustering was, however, claimed to be promising but no accuracy parameters were provided to support these results. Moreover, system performance in terms of time was not reported. The dataset used for evaluating the proposed technique was relatively small, and further investigations on larger dataset should make the research more authentic.

Agrawal et al. (2012) take a semantic approach to detect metamorphic variants of known malware families. High-level abstract signatures, in the form of graphs, are generated for known malware by performing static control flow analysis of the program binaries. For a given binary, super-block dominator graphs are generated for each function defined in the binary, and then a compound super-block dominator graph is produced by combining graphs of all functions. The resulting tree represents a semantic summary of the respective program binary. In order to test an unknown binary, its semantic signature graph is generated and compared with known signatures using normalized metric edit distances. The authors report to have achieved 86% accuracy in detecting unknown variants of known malware families. The authors are hopeful to achieve a higher accuracy by incorporating data flow analysis in generating the semantic signatures. The approach is specific for classifying unknown polymorphic or metamorphic variants of known malware families, and cannot be used to detect a malware belonging to a totally new family.

The solution proposed by Park et al. (2013) makes use of dynamic analysis to perform graph-based malware clustering. The system monitors system calls and arguments, and records a behavioral profile for each binary in terms of accessed kernel objects like processes, files, sockets, etc. The extracted profile is transformed into a directed graph, called the Kernel Object Behavior Graph (KOBG), in which nodes represent kernel objects and edges depict their inter-dependence through handles. The set of graphs for known malware is then clustered to identify malware families and a Weighted Common Behavior Graph (WCBG) is generated for each family which represents the behavior demonstrated by most members of the family. Then a unique subgraph is computed within WCBG, called the *hotpath*, which comprises the path exhibited by all members of the class. To check the nature of an unknown binary, its KOBG is compared with WCBG and hotpath graphs for each malware family. The unknown sample belongs to a specific malware family if its KOBG not only includes the family's hotpath graph but there also exists a particular level of similarity between its KOBG and the family's WCBG.

Evaluation of the system performance was performed on two different datasets containing 563 and 520 malware binaries respectively, belonging to 7 different families. WCBGs and hotpaths for each family were generated using the binaries in the training sets. Next, KOBG for each binary in the test set was matched against WCBGs and hotpaths of each family using the described algorithm. The detection rates for the first set peaked at 92% while those for the second data set reached 88%. The authors also tested the system for false positives, and used 100 executables from the Windows operating system for this purpose. The proposed algorithm showed 8% false positive rate for only one malware family while there were no false positives for the rest of the malware families. The paper does not state the time required by the system to produce the results, therefore system performance in terms of efficiency cannot be determined.

Tamersoy et al. (2014) have suggested the use of file relationships for labeling unknown files as benign or malware. The proposed solution is based on the assumption that, on a given machine, malware's relationship with other malware will be stronger than its links with benign files. The algorithm works on a dataset including benign and malware programs represented in terms of the files' location (machines). This representation is then used to infer co-occurrence strength between files using MinHashing. Next, the dataset is clustered into groups of files, called buckets, using Locality Sensitive Hashing on MinHash values. The files in a bucket have strong co-occurrence among each other. Files and buckets are then represented as nodes of a bi-partite graph and unknown files are labeled using Belief Propagation algorithm. Although the authors have reported a True Positive Rate of 0.996, the scheme has the pitfall of being dependent on data gathered from the community regarding the files' locations.

2.4 Visualization

Information visualization has been used in many domains of scientific research for presenting data to a human analyst in a concise and effective manner. Research in information visualization focuses on devising novel ways of representing and displaying complex data in an easily comprehensible graphical format. Application of information visualization techniques to the field of malware analysis and detection has several facets, as discussed below.

Yoo (2004) has proposed to use Self-Organizing Maps (SOMs) for visualizing viruses in Windows executable files. SOM is an Artificial Neural Network approach using unsupervised learning to map high dimensional data to a low dimensional space, suitable for visualization. The author postulates that training and visualizing an SOM on an infected executable file can show patterns, called virus masks, which are common to a whole virus family, not just representing an individual virus. The author proves the theory by testing various executables infected with viruses using an SOM library in Matlab. The vectors used as training input to the SOM consist of 4-byte values read directly from the binary file. The output is a 2-D unified distance matrix, which is displayed as a map consisting of hexagons. The hexagons represent input elements and are drawn in different colors, where a darker color represents a smaller distance (greater closeness) between neighboring elements. The results show that the executables infected with viruses of the same family exhibit similar patterns in visualization. However, the suggested technique is only used to find patterns, which is the training phase of the SOM. This work could be extended to actually classify unknown executables on the basis of trained data.

Trinius et al. employ different visualization techniques to present the behavior of a process, captured through a sandbox, to the malware analyst in a meaningful way (Trinius et al., 2009). Their approach uses treemaps for displaying the distribution of the operations performed by the program, and shows the order of operations through a thread graph. The original API call trace data obtained from the sandbox is very large so it is scaled down into categories or sections for easy visual comprehension. A treemap shows the sections of API calls as tiled rectangles, and the width of a rectangle depicts the number of API calls belonging to the particular section. Thread graphs represent the sequence of system calls made by all the threads of an application, and can be viewed in four levels of abstractions, level 4 being the most detailed. Combination of these two techniques provides an analyst with a clear view of a program's behavior in terms of the kind of operations it performs, as well as the frequency and order of operations. Furthermore, the authors extend the application of their technique to data files, postulating that a data file's behavior can be defined in terms of the operations performed by the application that processes the file. Put another way, when an infected data file is opened by an application, the tasks performed by the application will be different than in case of a benign or clean data file. The authors verified their approach on malware executables as well as infected data files with promising results. The ability to perform image-based clustering would definitely add value to the system.

A novel approach towards malware classification has been proposed by Nataraj et al. (2011). They treated malware binaries as images by simply plotting the bytes from the binary as pixels. Different malware programs belonging to the same family showed similar patterns, called textures in the image processing terms, when converted to images using the above method. Using this observation, the authors trained image processing based classifiers to identify unknown malware. Features from malware images were obtained using Gabor filtering technique, and k-NN classification with Euclidean distance was employed to perform the classification of malware images. When compared with the state of the art in malware classification techniques, the proposed approach not only showed accuracy comparable to the best reported accuracy, but also exhibited a marked increase in processing efficiency. Moreover, the authors claim that their method is robust against most obfuscation and packing techniques used by the malware developers for changing the malware binaries in order to hinder reverse-engineering and analysis attempts. However, the technique could fail if hackers modify their code significantly between malware families or use multiple packers or decryptors for obfuscation.

Saxe et al. (2012) describe a system for comparing malware samples to find behavioral similarities based on system calls. First the malware is executed in a virtual environment, and trace logs are generated for system calls made by the program related to file operations, system registry access and network communication. Next the trace log is partitioned into blocks, termed as semantic subsequences, using a boundary criterion based on the similarity of contiguous system calls. These two steps are performed for a number of available malware binaries, and the similarities between the semantic subsequences are determined pairwise on the basis of Jaccard Index, resulting in a similarity matrix. For validating their technique for finding similarity between malware samples, the authors performed graph-theoretic clustering of the trace log using the semantic subsequences as the features, and compared the results against Kaspersky[3] classification of the malware samples. For high values of the similarity threshold used in the clustering algorithm, the precision matched that of Kasperksy classification but recall decreased, while for lower similarity threshold, precision degraded in favor of recall. The authors did not report an optimal threshold value to keep both precision and recall within acceptable limits.

The proposed system offers a visualization-based interface with multiple views: the sequence visualization area displays semantic sequences in unique colors. Many such sequences are drawn for multiple programs, for efficient identification of similarities in actions performed by different programs. The sample similarity map is a 2-D grid, where each node in the map represents a unique sample. The position and color of each node is determined by applying Principal Component Analysis on the similarity matrix computed earlier. Similar nodes are drawn in same colors. Furthermore, programs which have a common semantic sequence in their traces are shown in clusters. Using the interface, an unknown sample can be visually linked to known malware, according to similarities in sequences and/or the objects (files, registry entries, network ports) accessed.

2.5 Ontology

An ontology describes concepts within a given domain, and asserts the relationships between these concepts through the notion of properties. Properties not only

[3]`http://www.kaspersky.com`

represent relationships, but are also used to indicate characteristics of concepts. Ontologies are expressed using a formalism which allows definition of concept and relationship hierarchies as well as facilitates to perform reasoning on the data such that implicit knowledge can be derived from the explicit information. Moreover, describing concepts, their properties, and relationships in a formal way allows knowledge to be shared among applications, and facilitates building a common vocabulary for the community (Gruber, 1995).

The use of ontology in the domain of information security was first suggested by Raskin and Nirenburg (2001). The authors emphasized the need for making use of natural language data sources to the domain of information security. They maintained that by capturing the common knowledge of terms used in the domain and the taxonomy by which these terms are related, the information security community can reap benefit of efficient sharing of knowledge about attack incidents. Furthermore, by formally and methodologically storing attack data in an ontology, prevention from and prediction of attacks could also be made possible. Although the proposal was at an abstract level yet it built the foundation for the use of ontologies in malware detection.

Similar thoughts were shared by Donner (2003), who also highlighted that the expanding field of information security needed an ontology which could facilitate efficient and effective exchange of information. In his article are mentioned not only sample concepts but also the relationships between these concepts. Contrary to Raskin et al., Donner focused only on the vocabulary aspects of the ontology, and did not propose its use in detection or prevention of malware attacks.

The work presented in (Fenz and Ekelhart, 2009; McIntire and Mundie, 2013) is representative of the efforts made by the research community towards the goal envisioned by Raskin and Donner as far as the sharing of malware knowledge, and making available a common vocabulary, is concerned. A review of security ontologies developed for the said purpose can be found in (Blanco et al., 2008); the research attempts involving use of ontologies for detection of malware are discussed here.

Undercoffer et al. list the reasoning power of ontology representation as one of the main reasons for using an ontology for their Intrusion Detection System (Undercoffer et al., 2003). In the proposed system, the ontology is represented using DAML+OIL, and the reasoning system employed for the system is DAMLJessKB. The concepts, properties and relationships are extracted from data pertaining to 4,000 attacks. Once the knowledge-base has been populated using concepts like Host, Attack, and Consequence, and relationships such as Host-VictimOf-Attack and Attack-ResultsIn-Consequence, DAMLJessKB extracts additional rules from the "chain of implication" of the original data. Queries can be formulated using both the concepts and the properties, and are executed by DAMLJessKB to retrieve any matching data. The queries can be made at different granularities i.e. levels of details about the incidents, and can identify a specific attack or all attacks belonging to a specific class, etc. Although this dissertation focuses on malware detection aspect of the cyber-security domain, but an example from the intrusion detection discipline points out that a similar ontology-based approach can also be adopted for malware detection.

Huang et al. presented an ontology-based malware behavioral detection system called TWMAN (Huang et al., 2010). The proposed system is claimed to be able to block Trojans and other unknown malware attacks. TWMAN executes known malware in a sandbox environment and captures changes in registry values, network connections, and file system observed during execution. These parameters serve as properties of the malware, and are stored in an ontology using Protégé, along with the malware hierarchy as concepts. An unknown sample can be classified by executing it in the sandbox and monitoring its behavior in terms of changes it makes to the parameters mentioned above. The ontology is updated with the observed properties of the unknown malware, and rule-based inference is then used to identify the malware class, if any, the malware sample belongs to. The presented work is just an initial proposal and does not report any performance metrics of the suggested system.

Huang et al. extended their work by incorporating fuzzy set theory with TWMAN

(Huang et al., 2011). While making minor changes to the process of behavioral model generation, the authors made use of Fuzzy Markup Language (FML) to create a fuzzy ontology for representing the behavior model. Instead of using the SWRL rules in Protégé, the authors incorporated fuzzy inference for performing reasoning. The fuzzy output Similarity (SI) was calculated on the basis of three fuzzy variables namely FileHash (FH), ConnectIP (CI) and FileChange (FC), using fuzzy rules built by human experts. The system performance was tested on 30 samples, for which the values of the fuzzy output SI were tabulated against the desired values. A graph of accuracy, precision and recall values is also given which plots these values as percentages against some threshold values, for which no reference could be found in the text. The system was able to score an accuracy of 82%, according to the reported graph.

The most recent update on the project makes a transition to the paradigm of soft computing through the use of type-2 fuzzy sets (T2FS) (Huang et al., 2014). The rationale behind using a soft computing mechanism is its ability to represent the inaccuracy, uncertainty and vagueness of the knowledge in the domain of malware detection. The system, called MiT, was evaluated by populating the ontology with 1360 known malware samples and then testing for 70 unknown samples. The authors report the accuracy of the system to be in the range 70%-80%. There is no evidence of the system's performance with respect to processing time, therefore it cannot be concluded if the proposed solution is scalable on a large dataset or not. The relatively low accuracy is suggestive of refinement of the inference mechanism, and selection of better features to be used as properties in the ontology.

2.6 Artificial Neural Networks

Artificial Neural Networks (ANNs) are inspired from the working of the nervous system (Rojas, 1996). Just like the way the neurons are inter-connected in the brain, the ANN consists of a network of processing elements (also called neurons), organized in layers in such a way that outputs of neurons in one layers are connected to inputs to the next layer neurons. The outputs may be weighted so that only the outputs of maximum weights inputs can activate, or fire, the next layer neurons. The ANN can be used in supervised as well as unsupervised learning scenarios for performing classification and clustering respectively. Due to its ability of performing pattern recognition, the ANN has been experimented with in the domain of malware detection. Some of the ANN based malware detection techniques proposed in the literature are discussed below.

The first attempt to practically apply principles borrowed from biological domain to the field of computer security can be attributed to Kephart et al. (1995). More specifically, two concepts were adopted from the biological area: neural networks and immune system. The former was used to implement a generic virus detector for known viruses, while the latter was adopted to develop an adaptive immune system for detecting unknown viruses. Three-byte long sequences from boot sectors were used in the prototype system as feature for training the classifier, and the prototype was evaluated over 5000 artificial boot sectors. The authors reported a detection rate of 90% with a low false positive rate of only 0.02%. The proposed system worked well for a class of viruses of that time but further evaluation would be needed to see the dynamic analysis techniques introduced later.

Stopel et al. used dynamic features with ANN for real-time worm detection (Stopel et al., 2006). Activities related to network usage, and processor and object properties were tracked and as a result 68 dynamic features were used in the proposed solution. Seven known worms were injected into computer systems and the activity on the systems was separately monitored for each worm monitored to obtain behavioral data pertaining to the aforementioned aspects. The authors report an average classification accuracy of 99.96%, 99.89% and 99.66% for the ANN, Decision Tree and *k*-NN respectively, for classifying known worms. The ANN was also tested for its ability to perform detection of unknown worms, and its detection rate was found to be 95%. ANN also performed faster in classification step, although it was slower in the training phase. Considering the small number of worms used in the experiment, therefore for more authentic results there is need for validating the proposed scheme on a larger dataset, containing multiple types and families of malware.

Dahl et al. (2013) have applied neural networks for large scale malware detection. Their scheme used dynamic features including strings found in memory region occupied by the program, tri-grams of API calls, and unique combinations of API calls and their parameters. First feature selection and dimensionality reduction steps were performed on the feature vector, and then a neural network classifier was trained on a dataset containing 2.6 million samples. Neural networks with different number of layers were trained, the single-layer neural network showing best performance with an average two-class misclassification error rate of 0.49%, at 0.83% false positive and 0.35% false negative rate. According to (Kephart et al., 1995), it is critical to keep the false positives low in security applications. In the scheme proposed by Dahl et al., the best single neural network achieved a false positive rate of 0.01% but with a false negative rate of 25%.

2.7 Genetic Algorithms

Genetic Algorithm (GA) is another computing technique based on a biological phenomenon (Mitchell, 1998). GA's mimic the natural selection process observed in biological evolution, and are used to solve optimization problems. Another major application of GA's is to find an optimal solution in a solution space where exhaustive search is infeasible. In such scenarios, GA's create a set of random solutions, called a population, and make use of a fitness function to evaluate them; a new population of candidate solutions is created as a result of mutation and crossover operations on the best solutions in the previous population. The process continues until an acceptable solution is reached. Following are a few research activities employing GA in the domain of malware detection.

Mehdi et al. have used Genetic Algorithm in their endeavor, called IMAD, to detect in-execution malware (Mehdi et al., 2009). The detection is performed by a classifier which uses n-grams of system calls made by the malware during execution; the GA is used for tuning of certain parameters involved in the detection process. The individual n-grams are assigned "goodness values" depending on the nature of the program which created them; n-grams observed during execution of benign programs are assigned the goodness of +1, those specific to malware are given -1, and the neutral group of n-grams (appearing during execution of both malware and benign programs) are assigned dynamically by the GA. When an unknown program is under observation, it is evaluated in terms of an "impression coefficient" which is calculated on the basis of the goodness values of the n-grams of generated system calls. If the coefficient rises above an upper threshold then the sample is declared benign; it is labeled as malware if the coefficient drops below a lower threshold. In the third case the program is allowed to execute further. The values of the impression coefficient, and the upper and lower bounds are also calculated by the genetic optimizer which uses a fitness function based on several parameters such as false positives and false negatives observed so far, etc. The efficiency of IMAD on a dataset of 100 samples was evaluated against SVM, C4.5, RIPPER and Naïve Bayes classifiers on the same dataset using WEKA. The authors report IMAD to have performed equal to RIPPER in case of 6-grams, surpassing all other classifiers, with a detection accuracy of 77% with zero false alarms. However the proposed system is only evaluated for a small dataset, and is susceptible to dynamic evasion techniques employed by malware developers such as detection of virtual environment. Furthermore, a revision of features may result in better accuracy.

Another research effort involving the use of GA is presented by Kim et al. (2010). Aimed at detecting polymorphic script viruses, the proposed technique first converts the code (script) into a code containing only unit statements. This semantically equivalent code is then represented as a dependency graph, followed by graph reduction to remove any redundant vertices. To check an unknown script, its reduced dependency graph is heuristically matched against all the precomputed graphs for known scripts. If a match is found within a specific threshold, then the script is considered to be a virus or a polymorphic variant of a virus. If the heuristics do not find a match then sub-graph isomorphism is applied in order to find graph similarity, and this is where GA comes into play. For comparing two graphs for isomorphism, one of the graphs is kept fixed and the other is mutated by rearranging its vertices and corresponding edges. Each such arrangement is a chromosome (potential solution). A combination of random and sequential solutions is generated for each population, and a fitness function is used to evaluate the solution. The GA stops if the difference between the generated solution and the target graph is below a certain threshold, or a maximum number of populations have been created without a match. The proposed technique is compared with 41 anti-virus engines in terms of detection accuracy and computation time. Although GA performs slower than the anti-virus software, its detection rates are encouraging because it captured more polymorphic variants of the 5 script viruses than the anti-virus software. However, the proposed system has been designed with a focus towards polymorphic script viruses only; an extension to make the work applicable to a wider variety of malware is therefore required.

2.8 Hidden Markov Models

A Markov model represents a non-deterministic system, comprising of different states such that there is a probability associated with transitions between states. At any given state, the next state is decided on the basis of these transition probabilities. A Hidden Markov Model (HMM) is a variation of the basic model in which the states are hidden; only the symbols that are associated with each state are visible. As in the case of state transitions, symbol probabilities determine which symbol is more likely to be output by a given state. Out of the three problems associated with the HMMs (Rabiner and Juang, 1986), problems 1 and 3 can be readily applied to the domain of malware detection. The first problem determines, given an HMM and an observation sequence, the probability that the sequence is generated by the HMM. In case of malware detection, one needs to find the probability that a given behavioral pattern is generated by an HMM representing the malware behavior. Problem 3 aims at determining the parameters (state transition and symbol probabilities) of an HMM. This step is analogous to the learning phase in machine learning paradigm, and is performed to train the HMM using observation sequences for known malware behaviors.

Probably the first use of HMM in the cyber-security domain was suggested by Warrender et al. (1999), who presented a comparison of HMM's performance with three other models for intrusion detection. The models were created using system call logs for certain benign programs, and represented "normal" system behavior. The authors reported that HMM was better than the other models in terms of accuracy, but with high computational cost.

In the malware detection domain, Wong and Stamp used HMM for detecting metamorphic viruses. In this static approach, HMMs were trained using opcode sequences from disassembled metamorphic programs, which were created from a virus construction kit. For each sample in a test dataset, which consisted of malware created from the same construction kit in addition to other malware and benign programs, a log likelihood score was computed against the trained HMM. The metamorphic variants of the same viruses obtained a high likelihood score as compared to other variants. This result proved that HMM could be effective for metamorphic malware detection. The authors used a relatively small number of hidden states for their experiments (2-6), as opposed to Warrender et al. (1999) who suggested that the number of states in an HMM for malware detection may be roughly equal to the number of distinct system calls. Following Warrender et al.'s guidelines and modeling the system with the number of states equal to the number of distinct opcodes may reveal interesting results. Furthermore, the authors have conducted the tests on a very small scale and their methodology involves human involvement in setting the thresholds.

Borrowing the idea of sequence alignment from the bioinformatics domain, Attaluri et al. have applied the Profile HMM (PHMM) to detect metamorphic viruses (Attaluri et al., 2009). PHMMs are a special kind of HMMs which take into account the relative position of symbols within the observation sequences while scoring an unknown sample. As the first step, multiple observation sequences (opcode sequences) for different metamorphic variants are combined into a multiple sequence alignment (MSA) structure which forms a general representation of all the constituent sequences. The MSA is used to generate a PHMM with additional *Insert* and *Delete* states, and associated transition and symbol probabilities. An unknown opcode sequence can then be scored against the PHMM for determining its likelihood of being malware or benign. The authors performed tests on three groups of programs, but the performance figures for only two successful tests are reported. The unsatisfactory results for the third group call for fine tuning of the proposed scheme.

In another PHMM-based approach proposed by Ravi et al. (2013), system calls have been used as observation sequences. Randomly selected malware samples from each family are used to create a multiple sequence alignment structure which, in turn, is used to train a PHMM representing that particular malware family. Behavioral reports for unknown malware samples (after being converted into sequence files) are scored against PHMMs for all malware families, and the normalized score vectors are used for clustering similar malware together. In order to use a sequence alignment tool which was specifically made for bioinformatics domain, the authors had to significantly downsize the number of system calls. This factor could adversely affect the performance of the proposed method.

Austin et al. experimented with HMMs in order to gain an insight into the semantics of hidden states (Austin et al., 2013). The proposed technique uses opcodes as features. By varying the number of hidden states in the HMMs, the authors identified the most significant opcodes and the associated probabilities. Termed by the authors as "dueling HMM strategy", the proposed technique can perform malware detection using two sets of HMMs. The first set is trained over features of benign programs compiled through different compilers, and the second set is modeled for known malware families. Given an observation sequence for an unknown sample, the HMM which yields the greatest probability of generating the given observation determines the nature of the sample. Testing the theory on metamorphic viruses and on code generated through virus construction kits revealed 87% detection rate which is good but still has room for improvement.

In a recent technique, Annachhatre et al. (2014) have used HMM for the purpose of malware clustering. They train the HMM using opcode sequences of programs generated by various compilers and virus generators. Opcode sequences of unknown malware are scored against the trained models, and these scores are used as distance metric in k-means clustering algorithm. The authors claim that their method results in a satisfactory clustering of malware samples. However, they suggest that results can be improved if their technique is extended to include malware specific HMMs. The malware dataset used for the experiment is unbalanced in the sense that bulk of the samples belongs to just three families out of 18, and this fact could have an effect on the results.

Damodaran et al. (2015) have presented a comparison of using static, dynamic and hybrid features for distinguishing malware from benign programs using HMM. In their experiments, the authors used four possible combinations of static and dynamic features for training and scoring HMMs, such as training on static/scoring on static, training on dynamic/scoring on dynamic features etc. Two sets of such experiments were performed using API calls and opcodes as features, and two different metrics (area under ROC and Precision-Recall curves) were used to compare the results. The experiments showed that the Dynamic/Dynamic combination performed better than the other three combinations for both types of features, especially in case of API calls. The size of the dataset used for this experiment was rather small (less than 800 malware and benign samples combined) and therefore a more exhaustive investigation using a larger dataset will be more beneficial for the malware research community. Furthermore, the experiments were performed with an aim to compare different types of features using HMMs and hence a malware classification scheme was not proposed as such.

The following criteria have been inferred through an analysis of reviewed literature for evaluating the previously proposed HMM-based malware analysis and classification methods:

1. **Feature used**: Dynamic features have been generally considered to be more discriminative as compared to static features

2. **Performance**: The performance figure depicts the effectiveness of the proposed approach.

3. **Dataset size**: Large dataset enhances the authenticity of the results.

4. **Comparison**: Has the proposed scheme been compared with other such methods? Effectiveness of the proposed scheme cannot be ascertained unless it has been evaluated against some other benchmark method.

5. **States analysis**: Has the proposed research analyzed the hidden HMM states in any way, especially with respect to their impact on performance?

6. **Efficiency analysis**: Has the proposed method been evaluated for efficiency? Is it scalable?

An evaluation of the HMM-based approaches reviewed herein shows that there is a dearth of research efforts with a reasonable score against the criteria described above. While use of both static and dynamic kinds of features has been investigated by most of the previously proposed solutions, the malware classification techniques based on Hidden Markov Models have mostly relied on static features. Most of the previous HMM-based malware analysis research has exploited sequences of opcodes for training the models, but use of system calls has also been observed in schemes based on Profile HMMs. It is therefore an open research area to further evaluate Hidden Markov Models on dynamic features.

Use of small datasets and lack of comparison of proposed malware classification methods with existing method(s) has also been observed, and this also emphasizes the need for further research in the area of HMM-based malware classification. Another major research gap that has been identified in HMM based malware research is that a crucial HMM parameter, the number of hidden states, has not been used with due consideration. Literature (Cave and Neuwirth, 1980; Rabiner, 1989; Levinson, 1987) suggests that this parameter corresponds to some intrinsic, hidden structure within the data being modeled, and therefore should be set after considering the properties of data. Instead, when modeling HMMs for malware analysis, researchers have opted for a trial and error approach, trying out different values for this parameter and adhering to one which results in the best performance. There is, therefore, a need for a study on the impact of this important HMM parameter on performance of HMM based malware classification schemes. Consequently, it is a requirement that HMM based malware classification process may be critically analyzed in order to find the optimum number of hidden HMM states for the malware classification problem. Furthermore, it is generally accepted that training an HMM is a computationally costly process but this factor has not been studied from practical perspectives. Performing an efficiency analysis of HMM based malware classification with an aim to evaluate its feasibility for realistic application is therefore in order.

2.9 Summary

Considerable research work has been done in the field of malware analysis and detection in the last two decades, and the survey presented here has only touched the representative few of the approaches found in the literature. The aim has been to include a variety of recently proposed solutions, based on ideas from a diverse range of scientific knowledge. The strengths and weaknesses of the surveyed techniques have been identified individually; a comprehensive summary is provided here as a ready reference.

Arguably the most notable observation from the reviewed literature is the high performance being reported for the proposed methods in terms of accuracy and effectiveness. The statistics show that malware is spreading with a threatening pace and consequently damage of billions of dollars is being faced by the world's economy, despite all the highly accurate malware detection and classification schemes seen in the literature. This situation not only opens the doors for further research in the domain of malware analysis, but also raises questions pertaining to evaluation of the proposed malware detection and classification methods. It can be seen that the proposed methods have been evaluated on specific, separate data and therefore their effectiveness against competing schemes can not be usually ascertained. Furthermore, other issues such as using small sized datasets and ignoring the efficiency aspects are also noticed.

Another key observation that can be made from the survey is that bulk of the efforts in malware analysis field has its roots in machine learning. Use of supervised learning paradigms of classification, ANN and HMM, and the unsupervised clustering mechanism have shown to be quite effective in identifying malware from benign programs and from other malware. The techniques proposed in these categories have shown by far the best results, and the accuracy figures convey the efficacy of the machine learning techniques.

Probably one of the main issues with machine learning techniques is the selection of useful features from a large feature set. For this reason, dimensionality reduction and feature extraction methods may be required prior to application of a machine learning algorithm. This causes additional computational overhead for such techniques. Furthermore, these schemes generally use fixed length feature vectors. For sequences of varying length (as in case of features extracted from behavioral reports) one needs to formulate a method to create a fixed length vector from the sequence. This transformation, if not done properly, may result in loss of valuable behavioral characteristics. Other, more general problems with machine learning based techniques discussed above include the computational cost of learning, high false positive rates, use of small dataset for evaluation, human involvement in selection of parameters used in the algorithms, and mostly reliance on static features.

Study of the solutions coming from visualization and ontology domains reveals the scale of knowledge diffusion in the malware analysis field. Although the volume of knowledge diffusion is not very significant, yet it is a clear indication of the fact that the scientific community is putting a great effort in finding an efficient solution for the malware problem by borrowing ideas from the vast scientific knowledge. The proposed techniques from the ontology domain do not show high accuracies, and visualization-based methods are primarily for manual inspection of malware, hence unsuitable for large scale classification.

Application of concepts from mathematics, such as graph theory, has also shown promising results. Both static and dynamic types of features have been investigated by researchers but there seems to be lack of efforts utilizing a hybrid set of features for representing a program as a graph. Furthermore, solutions based on graph isomorphism and graph matching are computationally expensive therefore approximation strategies have been applied in order to use these concepts for malware classification. The overall performance figures are not at par with other techniques.

A general problem observed in the reviewed research is that the research community generally does not adhere to a common metric for reporting the performance of their proposed methods and multiple parameters such as precision, recall, coverage, accuracy, quality (defined as the product of precision and recall) have been used by different researchers. This makes the task of evaluation and comparison among the research efforts difficult.

Table 2.1 summarizes the reviewed literature. A comparison of previously proposed HMM-based techniques for malware analysis and classification against the criteria described above is provided in Table 2.2.

Table 2.1: Summary of reviewed literature

S.No	**Reference**	**Data source**	**Methodology**	**Metric**	**Value**
CLUSTERING					
1	(Bayer et al., 2009)	Dynamic - System calls	Locality Sensitive Hashing	Prec * Rec	0.959
2	(Rieck et al., 2011)	Dynamic - Instructions q-grams	Prototype-based, complete linkage, hierarchical	F-measure	0.95
3	(Ye et al., 2010)	Static - Instruction (sequences, frequencies)	Hierarchical + k-medoids	Precision	0.9569
4	(Chandramohan et al., 2012)	Dynamic - System calls	Prototype-based, complete linkage, hierarchical	F-measure	0.948
5	(Hu and Shin, 2013)	Hybrid - frequencies of opcode n-grams, n-grams of system calls	Ensemble of prototype-based clusters	Precision$\uparrow$, Coverage$\uparrow$	15.5%, 46.5%
6	(Qiao et al., 2014)	Dynamic - API calls	Prototype-based, complete linkage, hierarchical	F-measure	90.9%
CLASSIFICATION					
7	(Schultz et al., 2001)	Static - Bytes, GNU strings, DLL	Naïve Bayes, Multi-Naïve Bayes, RIPPER	Accuracy	97.1%
8	(Rieck et al., 2008)	Dynamic - API calls	SVM	Accuracy	88.0%
	(Rieck et al., 2011)	Dynamic - Instructions q-grams	Prototype-based	F-measure	0.989
9	(Santos et al., 2013)	Static - opcode sequence frequency	DT, SVM, k-NN	Accuracy	95.9%
10	(Islam et al., 2013)	Hybrid - Function length, string count, API call frequency	SVM, IB1, DT, RF	Accuracy	97.06%
11	(Karampatziakis et al., 2013)	Hybrid - File features, System calls (parameters, trigrams), strings in memory	Logistic regression	FPR, FNR	0.2%, 15.2%
GRAPH THEORY					
12	(Hu et al., 2009)	Static - Function calls	Two-level indexing	Detection rate	90.0%
13	(Park et al., 2010)	Dynamic - System calls dependence graph	Feedback clustering	Dissimilarity	-
14	(Kinable and Kostakis, 2011)	Static - Function calls	k-medoids, DBSCAN clustering	Not reported	-
15	(Agrawal et al., 2012)	Static - Control-flow	Normalized edit distance	Detection rate	86.0%
16	(Park et al., 2013)	Dynamic - System calls	Graph-based clustering, graph matching	Detection rate	92.0%
17	(Tamersoy et al., 2014)	Static - File relationships	LSH, Bi-partite graph mining	TP rate	0.9961

Table 2.1 Continued...

S.No	Reference	Data source	Methodology	Metric	Value
VISUALIZATION					
18	(Yoo, 2004)	Static - 4-byte sequences	SOM	Not reported	-
19	(Trinius et al., 2009)	Dynamic - API calls	Treemaps, thread graphs	Not reported	-
20	(Nataraj et al., 2011)	Static - Code bytes	Gabor filter, k-NN	Accuracy	98.0%
21	(Saxe et al., 2012)	Dynamic - System calls	Graph-based clustering	Prec, Rec	0.9, 0.7
ONTOLOGY					
22	(Undercoffer et al., 2003)	Dynamic	DAML+OIL	Not reported	-
23	(Huang et al., 2010)	Dynamic - Changes in registry, files, network connections made	Protege, SWRL	Not reported	-
24	(Huang et al., 2011)	Hybrid - Malware hash, changes in files, network connections made	Protege, FML, Fuzzy Inference	Accuracy	82.0%
25	(Huang et al., 2014)	Hybrid - Malware hash, changes in registry, files, connections made	Soft computing	Accuracy	70%-80%
ARTIFICIAL NEURAL NETWORKS					
26	(Kephart et al., 1995)	Static - Byte sequences	Single layer	Detection rate	90.0%
27	(Stopel et al., 2006)	Dynamic - Network usage, processor and object properties	Levenberg- Marquardt	Detection rate	99.96%
28	(Dahl et al., 2013)	Dynamic - Strings in memory, API calls (trigrams, parameters)	Single layer	FPR, FNR	0.01%, 25%
GENETIC ALGORITHMS					
29	(Mehdi et al., 2009)	Dynamic - n-grams of system calls	-	Accuracy	77.0%
30	(Kim and Moon, 2010)	Static - Script (code)	Dependency graph, sub-graph isomorphism	Accuracy	88.9%

Table 2.2: Comparison of HMM-based malware classification methods

S.No.	**Reference**	**Method**	**Feature**	**Performance (metric)**	**Dataset size**	**Comparison**	**States analysis**	**Efficiency analysis**
1	(Warrender et al., 1999)	Intrusion detection	Dynamic (Sys call)	96.9% (TPR)	92,000	✓		
2	(Wong & Stamp, 2006)	Thresholding	Static (opcode)	Not reported	200			
3	(Attaluri et al., 2009)	Profile HMMs	Static (opcode)	˜67% (Det. Rate)	280			
4	(Ravi et al., 2013)	Profile HMMs, clustering	Dynamic (Sys call)	0.964 (Acc.)	19,000			
5	(Austin et al., 2013)	Max. likelihood	Static (opcode)	87% (Det. Rate)	60		✓	
6	(Annachhatre et al., 2014)	Clustering	Static (opcode)	0.94 (AU-ROC)	8,119			
7	(Damodaran et al., 2015)	Comparison of likelihood scores	Static, dynamic, hybrid	N/A	785			

Chapter 3

Malware behavioral classification using HMM

This chapter aims at answering the question: How can HMM be used to classify malware on the basis of their behavior. Consequently, this chapter describes in detail the methodology to use Hidden Markov Model for behavioral classification of malware. First a brief introduction to Hidden Markov Models is provided, which is followed by a description of the data used in the experiments. A method of using HMM for malware classification is proposed and evaluated using the behavioral data. The chapter concludes with a comparative analysis of the proposed classification method as well as a comparison of the proposed method with the state-of-the-art.

3.1 Hidden Markov Model

This introduction to Hidden Markov Model is adopted from (Rabiner and Juang, 1986). Hidden Markov Model is a statistical analysis method widely used for pattern matching applications such as speech processing (Rabiner, 1989), behavior modeling (Kuge et al., 2000), protein sequencing (Krogh et al., 1994), and malware analysis (Wong and Stamp, 2006; Annachhatre et al., 2014), etc. A Markov Model represents a stochastic system as a non-deterministic state machine, in which probabilities are associated with transitions. From a given state, the process transitions to the next state according to these transition probabilities. A Hidden Markov Model is an extension of the basic Markov model, in which the states are hidden and the progress of the process is observed in terms of certain symbols which the process emits in each state. Similar to state transitions, certain symbol probabilities are linked with each state, reflecting which symbol is more likely to be emitted by a process in a given state. In addition, a set of initial probabilities represents the initial state distribution of the model.

An HMM having N states and M symbols can be represented by $\lambda = (A,\ B,\ \pi)$, where A is the $N \times N$ state transition matrix, B is the $N \times M$ symbol emission matrix and π is the $1 \times N$ initial state distribution matrix.

HMMs have associated with them three basic problems:

1. How to calculate the probability that a given observation sequence $\mathcal{O}$ is generated by a given model λ?
2. How to determine the most probable sequence of state transitions that produced a given observation sequence $\mathcal{O}$?
3. How to estimate the model parameters $(A,\ B,\ \pi)$ from a given observation sequence $\mathcal{O}$?

These HMM problems can be solved by using well-established algorithms specifically applicable to the HMM. To be able to solve a given problem using HMM, the problem needs to be expressed in terms of one or more of the above HMM problems, and then the corresponding algorithm(s) is applied to solve the problem.

For the case of malware classification, HMM problem 2 is less significant because it does not contribute towards training or testing phases of machine learning; however, it might be helpful in analyzing the inherent malware behavior in a given sequence of observations. For malware classification or detection, problems 3 and 1 need to be solved in that order. First the model parameters need to be estimated (analogous to training a model) using observation sequences, and then this model is applied to an unknown sequence to determine the probability of the sequence being generated by the model. For training the HMM an observation sequence $\mathcal{O}$ is required along with initial (usually randomized) π, A and B matrices. HMM model creation involves use of Baum-Welch algorithm (Rabiner and Juang, 1986) which iteratively adjusts the probabilities in the three matrices until a predetermined number of iterations has been performed or the results of successive iterations lie within a threshold. The resulting model can be thought of as representing the average of the training sequences.

The trained HMM model can then be used to test an observation sequence using what is called the forward-backward algorithm (Rabiner and Juang, 1986) which returns the probability, or likelihood, of the sequence being generated by the model. In other words, the observation sequences under test are awarded likelihood *scores* against a given model. For efficiency purposes, implementations of the forward-backward algorithms may return the log of likelihood score, resulting in all scores to be below zero. An observation sequence closely resembling those used for training the HMM will have a negative score closer to zero, while a sequence not having any resemblance with the training sequences may be awarded a score of $-\infty$. Since the likelihood scores are dependent on the length of the sequence under test, therefore a score is normalized by dividing it with the length of sequence, i-e, the number of observation symbols in the sequence.

In the context of the malware classification problem addressed herein, the observation sequence used for training the HMM is represented by the sequence of system calls generated by the malware during execution. The hidden states correspond to some intrinsic structure within the system call sequence which the HMM tries to model by estimating the transition and symbol emission probabilities. An HMM trained over system call sequences for samples of a particular family is considered as representing the behavior of that malware family. The system call sequence of an unknown sample can be tested against the HMM trained for a specific malware family to determine if the sample belongs to that family on the basis of the likelihood score returned by forward-backward algorithm.

3.2 The dataset

For performing experiments conducted during this research, the data has been borrowed from the dataset[1] used in Malheur project (Rieck et al., 2011). The dataset includes system call logs for numerous malware recorded through CWSandbox (Willems et al., 2007). The malware samples were allowed to execute for 5-7 seconds, but the actual execution time depended on the malware being analyzed.

[1]`https://www.sec.cs.tu-bs.de/data/malheur/`

That is the reason that lengths of behavioral traces are inconsistent. While recording malware's behavior, only a single execution path was monitored. However, all the processes and threads were analyzed and their behavior was recorded.

The system call logs are provided in various formats but for the sake of this research, MIST format (Trinius et al., 2009) has been used. The MIST format represents system calls in multiple degrees of detail using a simple text layout, and therefore allows easy extraction of system call logs for a specific granularity. A representation of the `load_dll` system call is shown in Figure 3.1.

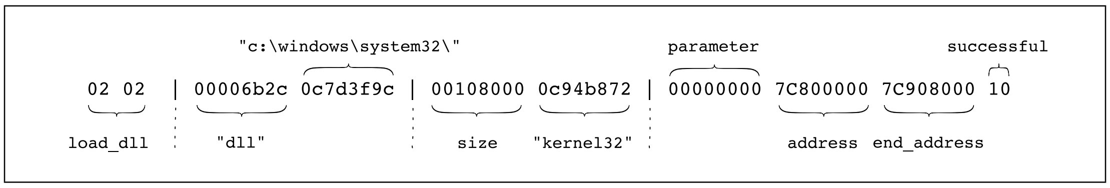


Figure 3.1: MIST representation of the `load_dll` system call (Trinius et al., 2009)

The MIST format categorizes the 120 monitored system calls into 20 categories according to the higher level operation being performed. System calls are represented by category ID, followed by the operation ID and arguments, if any. For the sake of this research, the system call arguments are ignored and only the higher level category and the individual function call are focused, corresponding to level 1 detail as named by Rieck et al. Furthermore, to simplify the representation of the (category, call) pairs as an observation sequence, each system call is represented as a number from 1 to 120. This is done by using the information provided in (Trinius et al., 2009) about the number of system calls in each category. The observation sequence, thus, becomes a sequence of numbers in the range from 1 to 120. These observations sequences representing malware behavior in terms of system calls are used for training the HMMs.

The full Malheur dataset contains system call logs for over 32,000 samples belonging to more than 400 malware families. For the presented work only those malware families have been selected which have more than 100 samples. This criterion was set in order to meet the requirement of sufficient number of samples for training the HMMs. Thirty seven such families were found (including the benign samples) which fulfilled this criterion, with the total number of samples being in excess of 29,000. Some of the malware families had too many samples therefore only 400 samples from each of those families were taken in order to make the dataset balanced. After the data reduction process there were 8,828 malware samples left. In addition, 300 benign samples grouped under NOTHINGFOUND class were also included in the dataset. Details of the selected malware families are provided in Table 3.1. The table shows, for each malware family, the number of samples (observation sequences), the minimum, maximum and average number of observations, and the cumulative length of the observation sequences used in the experiments.

A short description of these malware families is provided below. This information is obtained from different online resources such as Microsoft Malware Protection Center[2], ThreatExpert [3] and CNet[4].

Adultbrowser: This malware is of Dialer type. The malware of this family connects through a premium-rate telephone number to pornographic websites on the Internet and opens an HTML interface in the web browser.
Agent: It is a Trojan that connects to a remote website to download unwanted software. The content downloaded from the website possibly includes additional downloader Trojans, fake security tools such as anti-viruses, and any other kind of malicious software.
Allaple: A network worm which is multi-threaded and polymorphic. It spreads to other computers connected to the network and performs denial-of-service (DoS) attacks against targeted remote websites.
Autoit: Trojan whose main task is to download and install other malicious software, and make changes to the web browser settings.

[2]`www.microsoft.com/security/portal/threat/encyclopedia/search.aspx`
[3]`www.threatexpert.com/threats.aspx`
[4]`www.cnet.com/forums/discussions/virus-spyware-alerts-april-22-2009-340055/`

Table 3.1: Dataset specifications for comparison between proposed approaches

S. No.	Malware Family	Number of samples	Observation sequence length			
			Min.	Max.	Avg.	Combined
1	ADULTBROWSER	262	717	1011	739	193,618
2	AGENT	400	16	89,031	2931	1,172,304
3	ALLAPLE	400	15,171	17,848	15,813	6,325,191
4	AUTOIT	400	278	32,098	31,319	12,527,421
5	AUTORUN	172	36	51,147	5147	885,284
6	BIFROSE	152	20	24,859	626	95,152
7	BASUN	400	9044	88,223	56,866	22,746,471
8	BUZUS	142	15	51,697	2819	400,298
9	CASINO	140	296	1900	409	57,260
10	EJIK	168	236	250	242	40,656
11	FRAUDLOAD	133	15	54,882	5202	691,866
12	FRAUDPACK	232	16	30,291	5628	1,305,696
13	HUPIGON	227	36	52,505	956	217,012
14	KRAP	204	15	19,144	7572	1,544,688
15	LIPLER	380	1624	18,513	9639	3,662,820
16	LOOPER	209	3670	8294	5826	1,217,634
17	MAGANIA	201	36	27,931	1550	311,550
18	MAGICCASINO	174	123	132	127	22,098
19	OBFUSCATED	144	37	4218	1400	201,600
20	PATCHED	400	15	50,600	504	201,410
21	PODNUHA	308	18	141	136	41,888
22	POISON	375	36	62,886	531	199,125
23	RBOT	131	36	16,585	1135	148,685
24	REFROSO	178	18	9604	490	87,220
25	ROTATOR	300	1779	46,674	25,858	7,757,400
26	SALITY	173	21	16,708	2776	480,248
27	SMALL	106	17	29,455	3543	375,558
28	SPYGAMES	139	457	1123	558	77,562
29	SWIZZOR	400	268	4311	1745	697,945
30	TDSS	141	21	11,314	952	134,232
31	TEXEL	400	15	43,528	316	126,485
32	UDR	172	425	435	429	73,788
33	VB	400	64	63,626	1370	547,921
34	VIKING_DLL	158	334	2289	1972	311,576
35	VIRUT	400	15	42,263	42,263	460,011
36	ZBOT	107	16	12,855	4594	491,558
37	NOTHINGFOUND	300	1	87,462	14,524	4,357,300
	Total:	9128				65,833,520

Autorun: A worm which spreads through removable media storage. It copies autorun files into hard disk drives and also injects malicious code into system .dll files.

Basun: It is a worm that copies itself to other locations and changes Windows settings. It may download other malware.

Bifrose: This is a backdoor Trojan that connect to a remote IP address. It opens a network port and allows a remote attacker to access the computer and perform various actions through the opened ports.

Buzus: A malicious application which uses computer or network resources to completely replicate and spread itself.

Casino: This is a dangerous virus which may cause damage to the FAT. It has an additional feature that it asks the user to play a game: if the user wins the game, then the virus will not harm the FAT.

Ejik: This is an adware – a type of malware that is annoying but mostly harmless. The aim of this malware is to deliver advertising content to the user without their consent. It may be combined with spyware or tracking software to steal data.

Fraudload: This is a Trojan with additional downloader capabilities. It works by downloading unwanted files from a remote server, and then executing them after installation.

Fraudpack: This is a large family of malware which appear to be security tools but in fact are malicious programs.

Hupigon: This is a backdoor Trojan that steals personal information, such as user names and passwords used for online accounts for monitory transactions. They can also open a port for providing access to a hacker access.

Krap: A Trojan family that contains malware which tries to download other malicious program from websites.

Lipler: This family of malware tracks online users' habits by monitoring their web browsing history. This information is used to show pop-up ads.

Looper: This is a Trojan which gets installed when an unsuspecting user executes an unknown but believed to be safe program.

Magania: This is also a Trojan family which steals passwords and other personal and sensitive information. Another possible action by these malware is to install other unwanted applications.

Magiccasino: This is reported to be adware.

Obfuscated: A typical virus which replicates itself and infects other programs. It may also slow down the computer by stealing hard disk space and occupying large space in memory.

Patched: A Trojan that downloads malicious files and executes them locally.

Podnuha: Another Trojan family which contacts websites to download fake anti-malware tools

Poison: It is a backdoor Trojan which allows remote attackers to get unauthorized access to infected machines.

Rbot: Another backdoor family of malware which allow a hacker to control infected computers remotely through IRC channels. These backdoors may also steal information and spread to vulnerable computers.

Refroso: This is a worm family which halts Windows Security Center. It then spreads to other computers across the network by exploiting an unpatched Windows vulnerability.

Rotator: A typical adware that displays unwanted pop-up advertisements.

Sality: A large malware family which is capable of stopping security applications from executing on a system, steal information, replicate itself to other computers on the network, etc.

Small: These are downloader Trojans with the property that they have a very small footprint (a few kilobytes only). Typical actions include running in the background and downloading other malicious content to the infected computer.

Spygames: No description found for this malware family

Swizzor: Trojan downloader which also displays backdoor functionality. The remote master can use the backdoor to download and execute malicious files and adware.

TDSS: Trojans that have the rootkit feature for hiding their presence on the infected system. They also open a backdoor for the remote controller to access and steal personal information.

Texel: A family of worms which spread copies of itself though removable media or network channels.

UDR: A Trojan which adds and changes Web browser cookies, and adds harmful files to the system directory.

VB: A typical backdoor Trojan that enables the attacker to gain unauthorized access t the system. It also demonstrates adware functionality.

Viking_dll: A virus that runs as a service and effect executable files. It may try to download other malicious content from the Web.

Virut: A virus that attaches itself with the infected file to execute and replicate itself. Malicious actions include stealing information, deleting sensitive system files, and opening a backdoor for remotely controller malicious activity.

Zbot: Commonly reported as a password stealer, this Trojan can also affect security settings of a system and turn off security mechanism such as firewalls, etc.

Nothingfound: The set of benign programs.

3.3 Methodology

Figure 3.2 depicts the proposed methodology for HMM-based malware classification. The process starts with the behavioral reports of different lengths, represented by system calls, for n malware samples belonging to m malware families. It should be noted that these m and n are different from M and N mentioned in Section 3.1 on HMM, and should not be confused with each other. The malware samples are input to HMM modeling process on family basis so that an HMM trained from samples of malware family F_1, for example, represents the average behavior of this particular family.

After m HMMs have been modeled, all the n malware samples are subjected to evaluation against each of the models, and the result is a set of n score vectors, each of length m, where a score vector SV_k represents the similarity scores of the k^{th} malware sample against all m HMMs. These score vectors are then used in different ways to classify a given malware sample, as described in Section 3.3.2.

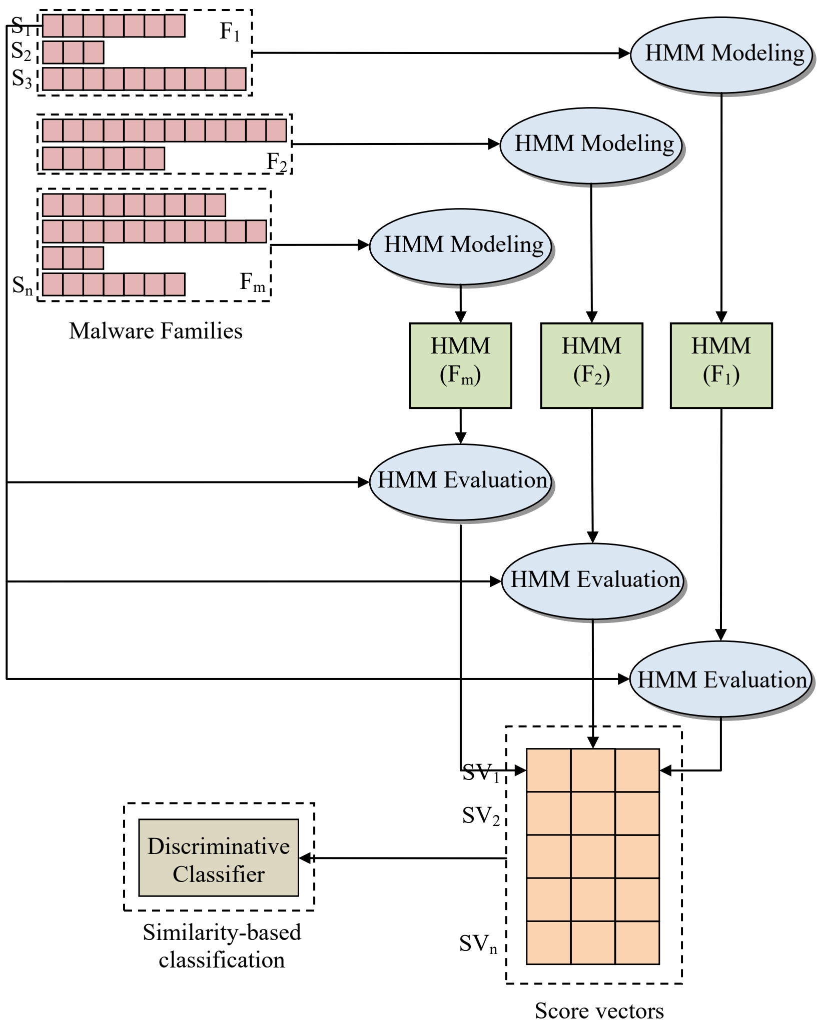


Figure 3.2: The HMM-based classification methodology

3.3.1 HMM training and evaluation

HMM model generation and testing was performed in Matlab using a third party HMM Toolbox[5]. Although HMM modeling can be performed in Matlab R2013a using the bundled statistical toolbox, the third-party toolbox was preferred because of availability of examples and ease of use. To further ascertain the viability of using the third-party toolbox, a comparative study was carried out between the built-in and third-party toolboxes using 1,510 malware samples belonging to six

[5]`http://www.cs.ubc.ca/~murphyk/Software/HMM/hmm_download.html`

malware families. The experiment was aimed at deciding which of these toolboxes is better suited for large scale experiments in terms of classification performance and time requirements. HMM modeling and evaluation was performed using both the toolboxes, and classification performance was observed in terms of F-measure. Matlab `tic` and `toc` commands were exploited to measure the time required by the two toolboxes for HMM modeling and evaluation.

The results of comparative study favored the decision to use the third-party Matlab toolbox for HMM modeling and evaluation. For the aforementioned small scale dataset, F-measure for malware classification using third-party toolbox turned out to be 0.899 as compared to F-measure of 0.883 achieved by using the Matlab's built-in toolbox. Similarly, the third-party toolbox was quicker than the Matlab's toolbox at performing HMM modeling and evaluation tasks as shown in Figure 3.3.

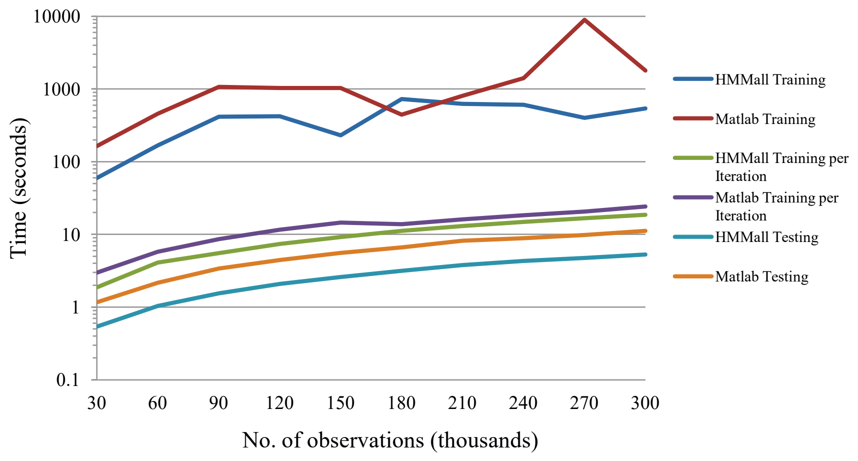


Figure 3.3: Time required for HMM modeling and evaluation by Matlab toolboxes

For a comprehensive evaluation of the proposed methodology, 5-fold cross validation was performed in the HMM training phase. For each family, five HMMs were trained such that for each HMM a different set comprising 80% of the behavioral reports for that family was used. During the testing phase, *all* the malware samples were evaluated against the five HMMs of each family. In this way, the proposed scheme was evaluated for three types of samples: those belonging to the same family and used in training (called known family samples), those belonging to the same family but not used in training the HMMS (termed as unknown family samples), and those belonging to other malware families (referred to as non-family samples). Furthermore, benign samples were also subjected to evaluation against the malware HMMs in order to assess the performance of the HMM-based classification techniques for those samples which are not malware.

For training the HMMs all of the malware families were used except the class representing benign files (NOTHINGFOUND). For training an HMM for a particular family, 80% of the observation sequences belonging to that family were fed sequentially to the Matlab routine which iteratively re-estimated the model parameters. The maximum number of iterations was set to 100, though in most cases the training stopped earlier because of successive log likelihood values being closer than the predefined threshold. The number of hidden states was set to 10 as a median between too few (2-6 used by (Wong and Stamp, 2006)) and too many (20, the number of categories of system calls used as observations). The number of observations was fixed to 120, corresponding to the number of unique system calls in the data.

In the testing step, each of the 8,828 samples was evaluated against all the trained HMMs. As discussed earlier, testing a sample sequence against an HMM returns a likelihood value (also termed as score) which determines how closely the sequence matches with the sequences that were used to train the HMM. The Matlab routine used for this experiment returns the log of likelihood values. Thus, testing a single sample against all the models produced a 36-long score vector of log likelihood values, and testing all the samples resulted in 8,828 such vectors representing the scores of all samples against all models. Since the likelihood scores fall in the range of 0 to 1, therefore the log values in the vectors were all negative, including $-\infty$ for cases where a sample had no resemblance with the modeled malware family.

3.3.2 Classifying the malware samples

According to categorization of sequence classification methods reported by Xing et al. (2010), HMM is representative of the model based sequence classification methods. However, Bicego et al. (2004) have suggested that HMM can also be used as a distance based sequence classification technique. They have identified two ways of using the HMM-generated score vectors to classify generic sequences, namely maximum likelihood and similarity-based methods which are used for the purpose of malware classification in this work. While the maximum likelihood method has been applied for the purpose of malware classification before (Austin et al., 2013), the use of similarity-based classification method has not been found in the literature. The two classification methods are described next.

3.3.2.1 Maximum likelihood classification

In this method of classification which belongs to model based sequence classification category according to Xing et al.'s categorization, a sample sequence is assigned a class label against whose model this sample obtains the highest likelihood score. In the case of malware classification, an unknown program that generates a system call log sequence s, is considered as belonging to the i^{th} malware family if the log likelihood score of s against the model $\lambda_{i \in N}$ is the highest among all N models. This can be represented as:

$$F(s) = \operatorname*{argmax}_{i} \log(P(s|\lambda_i)) \tag{3.1}$$

3.3.2.2 Similarity-based classification

The similarity-based classification scheme can be thought of as a distance based sequence classification method, where the 'distance' metric is represented by the log likelihood of a given sequence being generated by a given HMM. This method treats the set of score vectors obtained by evaluating sequences against HMMs as a feature space, termed as the *similarity space* by Bicego et al., which can then be used for training a discriminative classifier. The so trained classifier would then be able to classify an unknown sequence, given its similarity vector (the score vector), into one of the classes the classifier has been trained on. The rationale behind the name "similarity-based" is that the likelihood score obtained by evaluating a given sequence against an HMM represents its *similarity* (used as the distance metric) with the sequence(s) used for modeling the HMM.

The basic similarity-based approach for sequence classification trains HMMs for individual sequences. Since this research trains the HMMs on class basis, therefore the similarity-based approach could not be applied as such. Instead, a modified form of this approach was employed, also suggested by Bicego et al., in which the class-wise likelihood score vectors are used as feature vectors to train the discriminative classifier. WEKA (Garner et al., 1995) was used to train the various classifiers including Random Forest, J48 Decision Tree, Support Vector Machine (SVM), Naïve Bayes and k-NN over the labeled class-wise score vectors. Each of these classifiers was trained using different values for its key parameter as shown in Table 3.2. It was observed that the classifiers performed optimally with the default settings of these parameters in WEKA (shown in italics) with the exception of Random Forest classifier which showed a marginal improvement in performance as the number of tress grew. Random Forest proved to be one of the better classification algorithms, along with J48 Decision Tree and k-NN, for use with the malware score vectors in the similarity-based classification method as conveyed by Figure 3.4.

Random Forest classifier belongs to the "ensemble learning" paradigm within the domain of machine learning, in which results of multiple tree classifiers are aggregated to predict a sample's class (Liaw and Wiener, 2002). More specifically, it leverages the concept of bagging which corresponds to the process of creating tree classifiers on different samples of data, and then taking a majority vote for reaching a classification decision. The randomness is involved when splitting a node in a classification tree: Instead of choosing the best variable (predictor) from the whole set of variables for splitting a given node, the Random Forest classifier randomly creates a subset of predictors at each node and selects the best predictor

Table 3.2: Comparison of classifiers performance with respect to parameters

Classifier	Parameter	Value	F-Measure
Random Forest	Number of trees	2	0.84
		5	0.862
		10	*0.872*
		20	0.875
		50	0.876
J48	Confidence Factor	0.125	0.856
		0.25	*0.857*
		0.5	0.856
SVM	Kernel function	Sigmoid	0.018
		Radial basis	*0.808*
Naïve Bayes	Kernel estimator	*false*	*0.478*
		true	0.458
k-NN	k	*1*	*0.872*
		2	0.857
		5	0.836
		10	0.823

from the subset. Using this simple strategy, the Random Forest classifier is not only able to outperform other classification algorithms such as SVMs and Neural Networks (Liaw and Wiener, 2002), but it also deals effectively with the problem of overfitting (Breiman, 2001). For these reasons Random Forest was used in the experiments conducted for this research.

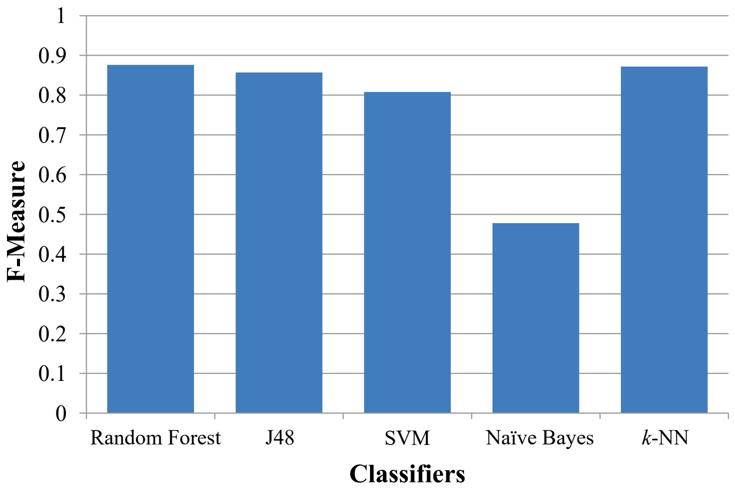


Figure 3.4: F-measures for similarity-based classification using different classifiers

3.3.3 Classifier validation

WEKA offers multiple choices of validation methods for testing how well a classifier is modeled on some given data. These validation methods include:

1. Using the same data for training and testing
2. Choosing separate datasets for training and testing
3. Splitting the dataset into training and testing partitions by a specific percentage
4. k-fold cross validation

The first method is obviously not desirable because it does not judge how generalized a trained classifier is. Sometimes the classifier memorizes the training set instead of learning from it, and therefore does not represent a generalization of the given instances. As a result such a classifier performs well on the data that has been used for training but does not classify previously unseen samples correctly (Bascil and Temurtas, 2011). The second method tests the classifier on unseen data, hence providing a better judgment about classifier performance, but requires different data files for training and testing. The third method is a simplification of the second such that it allows splitting the same data file into training and testing partitions without the need of separate files. The problem with the second and third evaluation methods is that they evaluate the classifier for just one set of training and test partitions. In such a case, the distribution of data instances in the two partitions might affect the classifier performance in a favorable or adverse manner. The k-fold cross validation method solves this problem by dividing the dataset in k partitions, and performing k classification sessions such that in every session, or fold, a separate set of k-1 partitions is used for training and the remaining partition is used for testing the classifier. In this way, each partition is used k-1 times as training partition and once as test partition, hence reducing the effects of any bias in the data. The classification results of k folds are then averaged to obtain the overall classification performance of the classifier over all the data. Due to this comprehensive nature of k-fold cross validation method, it was used to validate the Random Forest classifier in these experiments.

3.3.4 Evaluation parameters

Precision and *recall* are two widely used parameters for evaluating the performance of classification methods. Simply put, precision determines how many of the items labeled as the class C actually belong to C, and recall specifies how many items of a class C are correctly classified as C. The formulas for the two parameters are given as:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{3.2}$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{3.3}$$

To simplify the comparison of performance of different classification methods, precision and recall are usually combined into one parameter, called *F-measure*, defined as

$$\textit{F-measure} = 2 \times \frac{\textit{Precision} \times \textit{Recall}}{\textit{Precision} + \textit{Recall}} \tag{3.4}$$

An F-measure of 1 for a classification method means that the method was able to perfectly identify all the samples of each class. A low F-measure value, on the other hand, is result of either or both of the precision and recall being low, signifying the fact that certain items were misclassified by the classification method.

The malware classification schemes discussed in this paper were evaluated using F-measure.

3.3.5 The case of benign programs

Although the primary focus of this research was classification among malware families, the schemes under evaluation were also judged for their capability to handles benign files, i.e., normal program files known to be clean from any infection. For this purpose, the behavioral reports of 300 benign files grouped under NOTHINGFOUND family were evaluated against the 36 HMMs. The so obtained score vectors for benign samples were then subjected to both the maximum likelihood and similarity-based classification as described next.

Intuition suggests that a benign program's behavior should not match any of the malware behaviors. Consequently, the score vector of a benign program should ideally consist of only -∞ values. The maximum likelihood classification rule can, then, be modified to accommodate the benign samples as under:

$$F(s) = \begin{cases} \text{Benign}, & \text{if } \forall_{i \in 1..N} \log(P(s|\lambda_i)) = -\infty \\ \underset{i}{\operatorname{argmax}} \log(P(s|\lambda_i)), & \text{otherwise} \end{cases} \tag{3.5}$$

In the case of similarity-based classification, the score vectors for benign samples need to be tested against the classifier trained on the malware samples. It should be noted that the classifier trained on malware samples will only attempt to classify an unknown sample into a known malware family. In order to enable the classifier to identify benign samples, is should be trained on score vector(s) representing benign samples. As discussed above, the ideal score vector for a benign sample should only contain -∞. Therefore, the Random Forest classifier was trained on an additional score vector containing all values of -1000000, labeled NOTHINGFOUND. It may be noted that Since WEKA does not recognize the symbol -∞ returned by Matlab routine, therefore all instances of -∞ in the score vectors were replaced by the number -1000000, representing a very low likelihood, before using the score vectors in WEKA.

3.4 Results and discussion

3.4.1 Maximum likelihood classification

The maximum likelihood scheme for malware classification did not yield satisfactory results. Although for more than one third of the malware families the obtained F-measure was 0.99 or close, yet the overall average was 0.62.

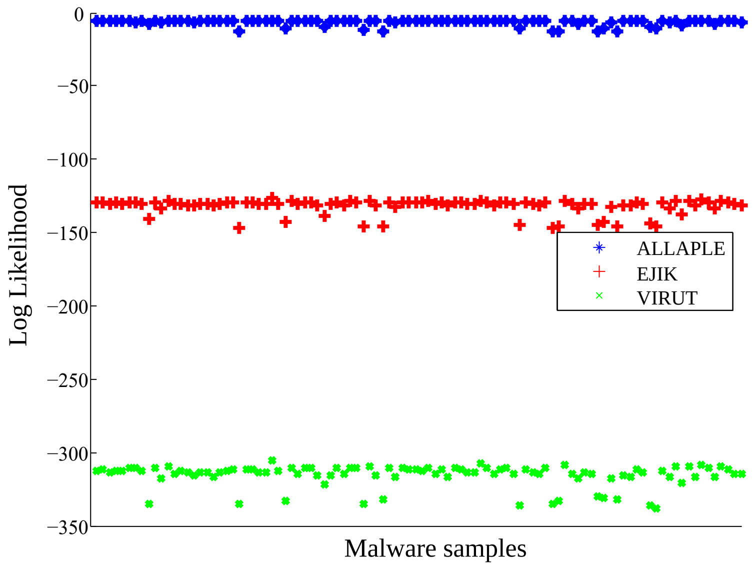


Figure 3.5: Log Likelihood scores of ALLAPLE samples against ALLAPLE, EJIK and VIRUT models

One high scoring family was ALLAPLE with an F-measure of 0.996. The scatter plot in Figure 3.5 shows the log likelihood scores obtained by ALLAPLE samples against ALLAPLE, EJIK and VIRUT family models. The higher the likelihood value for a sample against a model the greater the probability that the sample resembles the model. All values in the top portion of the scatter plot represent the likelihood scores of ALLAPLE samples against ALLAPLE model. For the other two models, the likelihood scores fall below, thereby making the classification decision clearly in favor of ALLAPLE family. This scatter diagram shows the likelihood scores of ALLAPLE samples against only three malware families; against all other models, these samples scored -∞, showing no resemblance with those families. The clear separation between the three data series is an indication that ALLAPLE family's behavior was consistent against all the models.

Some malware families with low F-measure were BIFROSE (0.10), BUZUS (0.11), PATCHED (0.12), and FRAUDLOAD (0.13). At first glance it looks counter-intuitive that a sample which has been used to train a particular HMM scores less than an unknown sample when tested against the model. Upon a closer inspection of some wrongly classified samples, such as in BIFROSE family, it was found

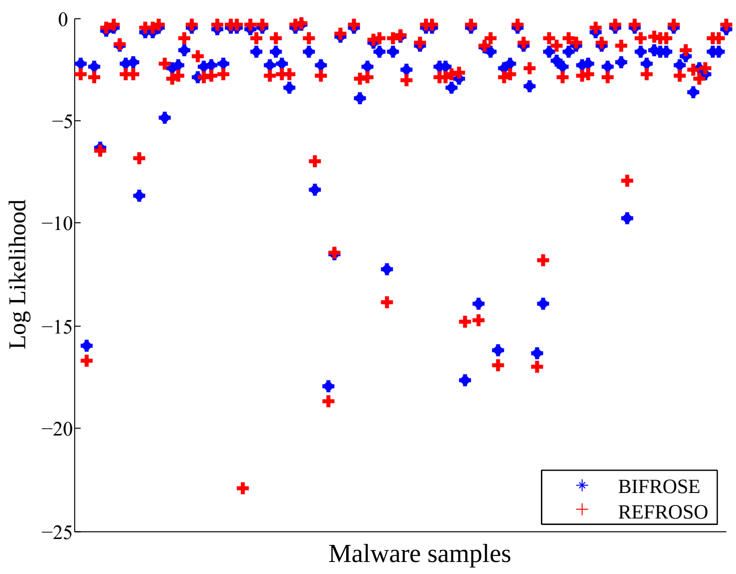


Figure 3.6: Log Likelihood scores of BIFROSE samples against BIFROSE and REFROSO models

that the error in misclassification was actually not indicative of a problem in the classification process. Instead, sometimes a significant portion of the behavioral profile of a sample can match that of a malware from another family.

The scatter plot of Figure 3.6 shows the likelihood scores for BIFROSE samples against BIFROSE and REFROSO family models. From the figure it is clear that the scores obtained by the BIFROSE samples against the two models fall within the same range, and therefore a correct classification decision is not possible on the basis of the maximum likelihood rule. As a result, many samples of BIFROSE family were classified as REFROSO by the maximum likelihood classification method. Interestingly, an online threat analysis report[6]verifies that different anti-virus software may identify a malware sample as belonging to BIFROSE or REFROSO families. This explains one reason for the misclassified samples while using the maximum likelihood approach on the score vectors.

The maximum likelihood classification rule failed completely when applied to the benign samples. Out of the 300 benign samples, none was correctly identified as benign; all these samples were labeled as different malware families. This was

[6]`http://www.threatexpert.com/report.aspx?md5=64a70c1bbc911c504dcfcb5917ece8c8`

because every benign program showed some behavioral similarity with one or more of malware families, and no benign sample was able to score -∞ against all the 36 models representing malware families in order to be identified as benign.

3.4.2 Similarity-based classification

The similarity-based classification approach produced much better results as shown in Figure 3.7 which plots the family-wise F-measure for the two classification methods. For roughly one-thirds of the malware families the F-measure remained well above 0.9, while the average F-measure across all the families was 0.87. Particular attention may be paid to the families BIFROSE, BUZUS and FRAUDLOAD for which the F-measure for similarity-based classification increased more than five times as compared to the maximum likelihood method. Other families such as RBOT and REFROSO also showed remarkable improvement as their F-measure values almost quadrupled for the similarity-based scheme. The average F-measure values for the two schemes are provided in Table 3.3.

The confusion matrices for the two classification schemes are shown in Figures 3.8(a) and 3.8(b) respectively. A comparison of the two figures shows the supremacy of the similarity-based classification method. Quite a few gray boxes, representing

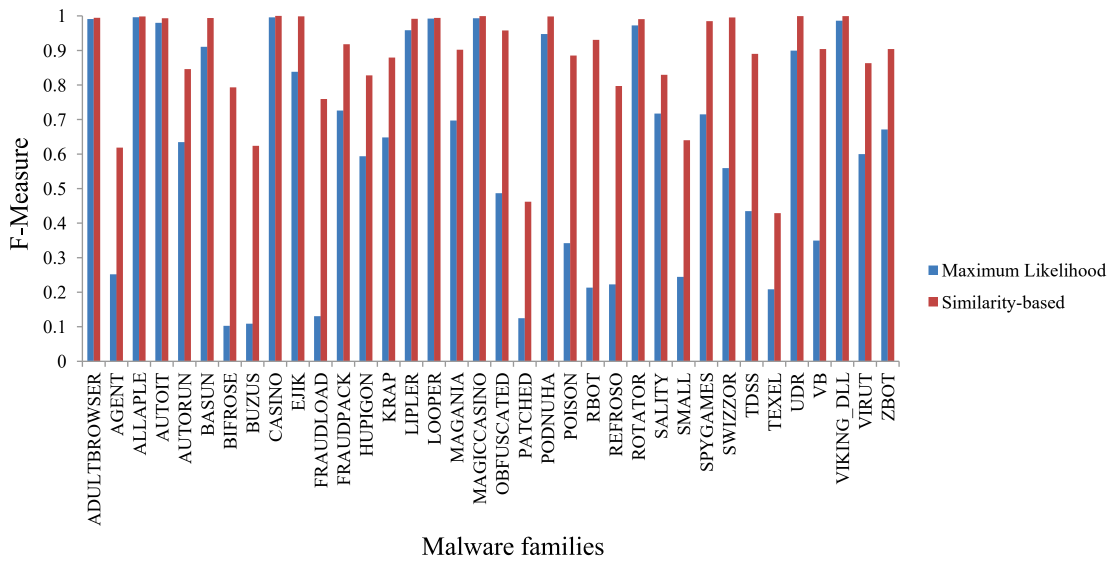
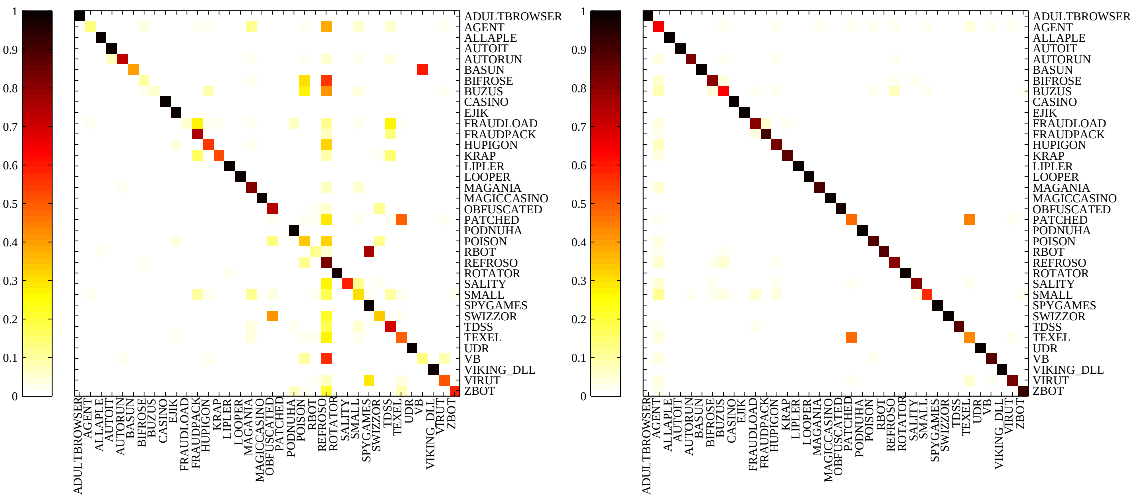


Figure 3.7: Comparison of F-measure for maximum likelihood and similarity-based classification

Table 3.3: Comparison of average F-measure values

Classification method	F-Measure	
	Malware samples	Benign samples
Maximum likelihood	0.62	0
Similarity-based	0.87	0.68



(a) Maximum likelihood classification (b) Similarity-based classification

Figure 3.8: Confusion matrices

the mis-classified samples, are seen on either sides of the diagonal in Figure 3.8(a). The column for REFROSO family is particularly noticeable, signifying the number of samples from other families classified as REFROSO. In comparison, the matrix in Figure 3.8(b) is mostly clear, asserting the fact that the Random Forest classifier was able to classify a majority of the samples correctly. The two dark boxes depict the samples of TEXEL and PATCHED families which the classifier was not able to label correctly. A malware analysis report[7] shows that different anti-virus scanners may also label a given malware sample as a TEXEL or PATCHED family malware, confirming the finding by the similarity-based classification that the two families share some common behavior.

3.4.3 Comparison with the state-of-the-art

Having observed the better performance of similarity-based scheme over the naïve maximum likelihood approach for malware classification, the next step was to

[7]`https://www.virustotal.com/en/file/6f737b428b0a753e1f542f5f021c1e97dfd0c2646956\9085753c810e83f8afd0/analysis`

compare it against the state-of-the-art. Since data used for this research had already been used by Rieck et al. (2011) for evaluating their malware classification and clustering scheme, therefore it was logical and convenient to compare the results of HMM-based malware classification method with theirs.

Rieck et al. have employed two different datasets in their work. First is the smaller dataset, called the *reference* dataset, which they used for fine tuning certain parameters, and second is the larger and more comprehensive dataset, termed as the *application* dataset, on which they performed detailed experiments. The authors have stated the classification performance of their scheme for the reference dataset only. For performing a comparison between the similarity-based classification method with their method on common grounds, the similarity-based classification was performed on the reference dataset also. Details of the reference dataset are provided in Table 3.4.

Rieck et al. have reported an F-measure of 0.981 for their classification method for known malware and 0.997 for unknown malware. The results for known malware indicate the performance of their scheme for classifying a known malware into its family, while for unknown malware the score represents how well the given malware sample is identified as belonging to any malware family. Since in their experiments half of the samples used by Rieck et al. were known and half were unknown, therefore the overall F-measure can be computed as average of the two reported figures, which turns out to be 0.989. The similarity-based scheme proposed in this research showed an F-measure of 0.994 for the same dataset. As previously explained in the methodology section, the similarity-based scheme handles the case of both known and unknown malware. The score vector for a given sample represents its similarity with all malware families; the sample is 'known' for just one malware family and it is unknown for all other families. The F-measure for the proposed scheme is thus comparable to Rieck et al.'s method, and it proves the point that HMM can be quite effectively used for classification of unknown malware into known malware families. It is again emphasized that primary focus of this research is *not* to highlight HMM as the best malware classification method;

Table 3.4: Dataset specifications for comparison with the state-of-the-art

S. No.	Malware Family	Number of samples	Observation sequence length			
			Min.	Max.	Avg.	Combined
1	ADULTBROWSER	262	717	1011	739	193,618
2	ALLAPLE	300	206	17,840	15,580	4,674,000
3	BANCOS	49	14,551	67,588	30,892	1,513,708
4	CASINO	140	296	1900	409	57,260
5	DORFDO	65	175	321	186	12,090
6	EJIK	168	236	250	242	40,656
7	FLYSTUDIO	33	134	29,533	1575	51,975
8	LDPINCH	43	36	226	146	6278
9	LOOPER	209	3670	8294	5826	1,217,634
10	MAGICCASINO	174	123	132	127	22,098
11	PODNUHA	300	82	141	138	41,400
12	POISON	26	36	860	277	7202
13	PORNDIALER	97	1327	3803	1433	139,001
14	RBOT	101	316	4591	375	37,875
15	ROTATOR	300	1779	46,674	25,858	7,757,400
16	SALITY	84	21	8293	2578	216,552
17	SPYGAMEs	139	457	1123	558	77,562
18	SWIZZOR	78	578	4301	4070	317,460
19	VAPSUP	45	1114	1136	1114	50,130
20	VIKING_DLL	158	334	2289	1972	311,576
21	VIKING_DZ	68	114	25,236	14,415	980,220
22	VIRUT	202	36	5930	376	75,952
23	WOIKOINER	50	9920	10,312	10,144	507,200
24	ZHELATIN	41	211	1849	1694	69,454
	Total:	3132				18,378,301

the aim is to evaluate how feasible it is to use HMM-based malware classification in a practical scenario. The current experiment yields that HMM does have the ability to classify known and unknown malware, while other aspects of HMM-based malware classification methods will be addressed in the next chapters.

One advantage of the similarity-based technique over Rieck et al's scheme is that HMM estimates the required parameters directly from the given data, while their technique depends on manual calibration of certain thresholds. The thresholds optimized for one dataset may not work as well for the new data added during the incremental analysis. On the other hand, HMMs have a basic drawback in the form of high computational cost of learning from sequences. The fact that model learning is performed much less frequently than model evaluation may favor use of HMM based methods because HMM model evaluation is a significantly less computationally intensive task than HMM modeling.

Chapter 4

Hidden Markov Model as Feature Extraction Method

Manifold growth of malware in the recent years has resulted in extensive research being conducted in the domain of malware analysis and detection, and theories from a wide variety of scientific knowledge domains have been applied to solve this problem. The algorithms from the machine learning paradigm have been particularly explored, and many feature extraction methods have been proposed in the literature for representing malware as feature vectors to be used in machine learning algorithms. In this chapter the similarity-based malware classification method, proposed in the previous chapter, is evaluated further by considering it as a feature based classification method according to Xing at el.'s categorization (Xing et al., 2010). Such methods of sequence classification represent variable length sequences as fixed length feature vectors by extracting representative features from the sequences, and employ conventional classifiers over the feature vectors for classification of unknown sequences. This chapter, thus, presents a comparison of HMM-based feature extraction method with several other feature extraction techniques found in the literature by first applying them on system call logs of real malware, and then evaluating them using Random Forest classifier.

As discussed above, machine learning based algorithms have gained special attention of the malware research community. The main reason behind using these techniques is that they enable detection of a previously unknown threat using models leaned from known malware. This is a key requirement in today's cyber world where millions of new malware are reported every day, and a large number of the new malware is obfuscated from existing malware (Elhadi et al., 2014).

The machine learning based techniques generally require a feature vector representation of malware, where a feature represents a particular malware attribute that can play a discriminatory role in the classification process. Extracting discriminatory attributes pertaining to malware and representing them in a way to be effectively used in a machine learning setting is a major challenge in the domain of malware analysis and detection. Termed as feature extraction in the machine learning paradigm, this process is a prerequisite to every malware detection technique proposed in the literature that employs a machine learning algorithm. When malware's behavior is represented as a *sequence*, use of feature extraction corresponds to a sequence classification technique termed as feature based classification (Xing et al., 2010).

In this chapter HMM is treated as a feature based classification method, and a comparison of HMM based classification method with eight other methods of extracting features from sequences is reported using a comprehensive dataset which contains system call logs of real malware. Specifically, sequences of system calls are transformed into fixed length feature vectors by applying different feature extraction methods, and these transformations are evaluated by performing malware classification using Random Forest classifier on the feature vectors. There is no gold standard dataset as such for performing a comparative evaluation of malware classification techniques, and all of the feature extraction techniques have been tested on different data. The main contribution of this work, therefore, is that it compares different feature extraction methods on the same data, hence providing an empirical comparative assessment of these methods. This research also adds value to the knowledge by evaluating a novel approach of combining the features extracted through different methods for the purpose of malware classification.

Liu et al. (2005) performed a similar study about ten years ago in which three feature extraction methods were addressed. Spanning the research of the last decade and earlier, th presented work covers a broader range of feature extraction methods. A recent comparison of feature extraction methods can be found in (Ranveer and Hiray, 2015) but it focuses mainly on a survey of such methods and falls short of evaluating them on some benchmark data.

This chapter proceeds as follows. Section 4.1 imparts some necessary background knowledge on the feature selection methods being compared in this paper, and provides examples of use for such methods from the literature. The methodology adopted for the experiments is explained in Section 4.2. Results are presented and discussed in Section 4.3.

4.1 Feature extraction methods for malware analysis

Machine learning algorithms learn the patterns from fixed length feature vectors, and therefore feature extraction is the first step before using these algorithms for malware analysis. For features that are in the form of sequences, such as sequences of code bytes, operation codes (opcodes), system calls or API calls etc., the creation of a representative feature vector is a non-trivial problem, and hence various feature extraction methods have been proposed in the literature for this task. Here the most widely used feature extraction and representation techniques applied on sequences are outlined. A finite set $\mathbb{S} = \{s_1, s_2, \ldots, s_n\}$ needs to be defined containing, in a specific order, all the unique symbols s_i allowed to make up a sequence. The set $\mathbb{S}$ may be considered as a term dictionary in the information retrieval terminology. Any arbitrary sequence, containing possibly repetitive occurrences of elements of $\mathbb{S}$ in any order, can then be represented as $S = (s_k)_{k \in \{1..|\mathbb{S}|\}}$.

Let us consider, as an example, the following sequences containing occurrences of three distinct elements a, b and c:

$$S_1 = (c, a, b, c, b, b, a, c, c, b, a)$$
$$S_2 = (b, a, c, c, a, a, c, b, a, c, a, c, b, b, c, c, a, b, b, a, c)$$
$$S_3 = (a, c, a, a, c)$$
$$S_4 = (b, c, c, b, a, a, b, a, c, c, b, c, c, a, a)$$

The term dictionary for the example dataset is represented by $\mathbb{S} = \{a, b, c\}$.

4.1.1 Binary feature extraction

The baseline method of extracting features from a sequence is to identify all the distinct elements found in the sequence. The sequence can then be represented as a binary vector of the same length as the term dictionary $\mathbb{S}$ such that each feature in the vector signifies the presence or absence of the corresponding dictionary term in the sequence. The resulting feature vector can be represented as $V_{Sb} = (b_{s1}, b_{s2}, \ldots, b_{sn})$, where b_{si} is 1 if S contains at least one instance of s_i and 0 otherwise, and n is the size of $\mathbb{S}$.

Using frequency feature extraction method, the given sequences of different lengths can be represented as fixed length feature vectors in the following way.

$$V_{S_1b} = (1, 1, 1)$$
$$V_{S_2b} = (1, 1, 1)$$
$$V_{S_3b} = (1, 0, 1)$$
$$V_{S_4b} = (1, 1, 1)$$

It may be noted that the arrangement of features in all the feature vectors *must* adhere to some predetermined sequence of distinct elements in the dataset (the term dictionary) which, in this example, is (a, b, c). The first element of a feature vector (the first *feature*) represents the presence or absence of the element a in the original sequence, the second feature depicts the presence or absence of the element b, and so on.

Examples from literature

Tian et al. (2010) proposed a binary feature technique for malware detection and classification using API calls. The authors also experimented with the frequency based methods on the same data but no improvement was observed over the binary representation. In a similar approach, Devesa et al. (2010) monitored API calls of malware and benign programs, and derived rules for extracting actions performed by the programs from API call logs. The term dictionary therefore consisted of the performed actions, and a binary feature extraction was performed.

Schultz et al. (2001) used three kinds of static features extracted from DLL related information, printable strings and binary code of malicious files. In case of the code bytes, all the two-byte words found in malware code were combined into the term dictionary, and binary feature representation was used for each malware sample. Following their footsteps, Kolter & Maloof (2004) created features using 4-grams of byte codes from executables. Since the feature space grows rapidly with increasing n, the authors used information gain to select 500 most representative 4-grams, and used them as binary features in various machine learning algorithms.

4.1.2 Frequency feature extraction

In this feature extraction method, the count of occurrence of a dictionary term in the sequence is used instead of just its presence or absence (Santos et al., 2013). Mathematically, this vector can be denoted as $V_{Sf} = (f_{s1}, f_{s2}, \ldots, f_{sn})$, where f_{si} is frequency of the i^{th} element of $\mathbb{S}$ in S.

Continuing with the above example, the frequency feature vectors representing the four sequences can be computed as follows:

$$V_{S_1f} = (3,4,4)$$
$$V_{S_2f} = (7,6,8)$$
$$V_{S_3f} = (3,0,2)$$
$$V_{S_4f} = (5,4,6)$$

Examples from literature

Lee et al. (1997) modeled each 7-gram of system calls observed during execution of normal and intrusive executions of Unix sendmail program as a binary feature in their intrusion detection scheme. Alazab et al. (2010) represented the API sequences with n-grams for values of n from 1 to 5. In order to keep the feature space to a manageable size, frequencies of all n-grams for a given n were computed for the whole dataset of malware and benign samples, and 100 most frequent n-grams were selected to be included in the feature vector. Altaher et al. (2011) extracted API calls from PE executables, and frequency of each API call was considered as a potential feature. All the API calls were then ranked according to their information gain, and frequencies of top 50 API calls constituted the final feature vector.

4.1.3 Frequency weight feature extraction

Frequency weighting methods such as *term frequency-inverse document frequency* (TF-IDF) have also been employed to generate feature vectors from sequences (Marian et al., 2012). TF-IDF is a statistic widely used in information retrieval and text mining domains for calculating frequency based weights for terms in a document in order to assess their relative significance.

The term frequency usually refers to the simple count of occurrences of a given term, or word, in a document. The inverse document frequency is computed by taking the logarithm of the total number of documents in the corpus divided by the number of documents in which the given word appears at least once. These two computational phenomena, when combined together, form a measure of importance of a word relative to other terms in the corpus. Given a term t, a document d and a corpus (collection of documents) D, TF-IDF can be mathematically represented as:

$$tfidf(t,d,D) = tf(t,d).idf(t,d,D) \tag{4.1}$$

where $tf(t,d) = f_{t,d}$ is the count of occurrences of term t in document d, and $idf(t,D)$ is given by Equation 4.2.

$$idf(t,d,D) = log\frac{N}{|\{d \in D : t \in d\}|} \tag{4.2}$$

where $N = |D|$ is the number of documents in the corpus.

In the above example, the elements of the sequences depict words whereas sequences represent documents; the corpus consists of the four sequences. The TF-IDF for the elements a, b, and c in sequence S_1 can be computed as follows:

$tfidf(\text{“}a\text{”}, S_1) = 3.log(\frac{4}{4}) = 0$
$tfidf(\text{“}b\text{”}, S_1) = 4.log(\frac{4}{3}) = 0.50$
$tfidf(\text{“}c\text{”}, S_1) = 4.log(\frac{4}{4}) = 0$

In a general form, frequency weighting vector for S can be represented as $V_{Sfw} = (fw_{s1}, fw_{s2}, \ldots, fw_{sn})$, where fw_{si} represents frequency weight for the i^{th} element of $\mathbb{S}$ as computed over S. For the given example, $V_{S_1fw} = (0, 0.50, 0)$. The frequency feature vectors for S_2 and S_3 can be computed similarly.

Examples from literature
A feature extraction method in the domain of intrusion detection is found in (Liao and Vemuri, 2002) in which the authors constructed fixed length feature vectors from system call sequences by computing TF-IDF as the weight for each unique system call in the corpus, and using this weight as a feature. Research reported in (Marian et al., 2012) made use of TF-IDF on frequencies of low-level kernel calls to represent kernel calls made by a program as a weight vector which was then used for cosine similarity computation. In recent works, Lin et al. (2015) suggested computing of TF-IDF within each malware family, instead of on the whole corpus, for feature selection.

4.1.4 Hidden Markov Model

Another method of extracting features from sequences is based on Hidden Markov Models (HMMs) (Bicego et al., 2004). Although the authors of the referenced text do not attribute their technique as a feature selection method, yet the process that they have adopted performs the exact task of converting a given sequence into a fixed length vector which can subsequently be used by a discriminative classifier for the classification of sequences. The same method has been applied in the similarity-based malware classification scheme presented in Chapter 3.

The technique presented in (Bicego et al., 2004) involves training one HMM for every class of sequences. Let us suppose the input sequences belong to and are labeled with m classes. Symbols in this case are represented by the individual system calls. Let us further assume the total number of sequences to be k. After the training of HMMs has been done, there are m HMMs representing different classes of sequences. The next step is to evaluate all k sequences against the m HMMs and for each sequence its likelihood score is recorded as a feature. In this way, a feature space consisting of k feature vectors is produced where each vector is of length m. In mathematical notation, the feature vector for a sequence S would be $V_{Shmm} = (l_{S,1}, l_{S,2}, \ldots, l_{S,m})$, where $l_{S,i}$ is the likelihood score obtained by evaluating S against i^{th} HMM.

Examples from literature

One of the few approaches employing Hidden Markov Model as a feature extraction method was proposed by Annachhatre et al. (2014) who used opcode sequences extracted from various compilers and virus generators to train HMMs. Opcode sequences from malware samples were then scored against the HMMs to generate the feature vectors which were then used for clustering of malware samples. As discussed earlier, a variation of this method was proposed as a part of this research which modeled malicious behavior instead of compiler behavior. In the proposed approach, malware behavior was represented as sequence of system calls which were used to train separate HMMs for different malware families. Known malware were evaluated against the malware family HMMs and the resulting score vectors were used to classify unknown malware using discriminative classifier.

4.2 Experiments

4.2.1 Feature set generation

Nine sets of feature vectors were generated against the system call sequences for comparative evaluation of the feature extraction methods, as described below. First, the feature extraction methods based on binary, frequency and frequency weighting factors were applied on system call unigrams of all the 8,828 sequences in the dataset. This resulted in three sets of feature vectors, namely unigram binary, unigram frequency, and unigram frequency weighting sets.

Since the sequential information in the input data is lost by applying the above mentioned feature extraction methods on unigrams, therefore use of bigrams is suggested in the literature to preserve the order of items (system calls in this case) in the input data. For this reason, three more feature sets were obtained by applying the binary, frequency, and frequency weighting feature extraction methods on bigrams of system calls. The term dictionary for the system call bigrams was populated with 2,745 unique bigrams extracted from the dataset, and therefore each of the feature vectors for the bigram schemes included 2,745 features.

Some of the reviewed schemes performed feature selection based on Information Gain. In order for this study to cover the feature selection aspect, feature ranking on the bigram frequency feature set was performed based on Information Gain using WEKA. The top 100 highest ranked (most significant) frequencies were then selected as features in the seventh feature set, which will be referred to as *bigram freq IG* set later in the document.

The eighth set of feature vectors was obtained by training 36 HMMs on system call sequences belonging to the corresponding malware families. Each malware sample was then scored against each of the HMMs to obtain the feature vector composed of likelihood scores.

Another experiment was carried out to judge the impact of combining feature vectors obtained through different feature extraction methods. As a trial, the HMM based feature vectors were truncated with *bigram freq IG* feature vectors to obtain the ninth and final feature set.

4.2.2 Feature set evaluation

For evaluating the nine sets of feature vectors obtained by applying various feature extraction methods, these sets were subjected to classification using Random Forest classifier in WEKA version 3.7. Random Forest classifier was selected after

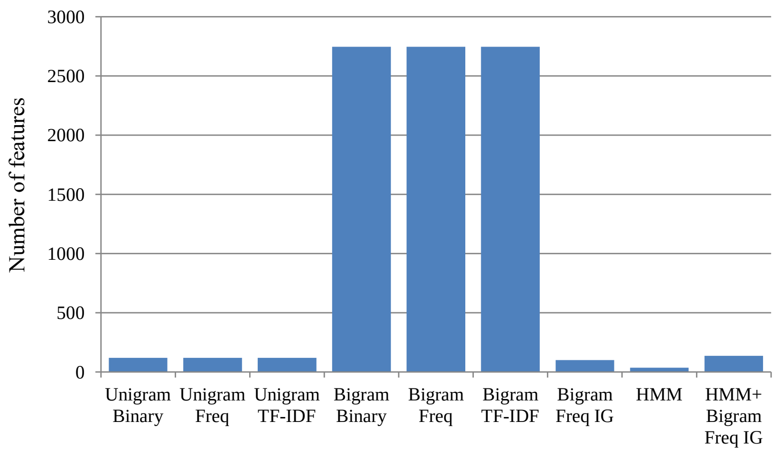


Figure 4.1: Number of features extracted through different feature extraction schemes

comparing its performance against various other classification algorithms including J48 Decision Tree, *k*-NN, Naïve Bayes, and Support Vector Machine (SVM) as described earlier in this document.

4.3 Results and discussion

Figure 4.1 shows the number of features extracted by the nine schemes discussed above. The HMM based feature extraction method proved to be the most concise in terms of the feature count with only 36 features representing arbitrarily long sequences of system calls. On the other extreme, the number of bigram features exceeded 2,700.

The graph of Figure 4.2 shows the effectiveness of feature extraction methods on Random Forest classifier in terms of average F-measure across all the families. Subsequently in this document, the term F-measure represents the averaged value.

To analyze the overall classifier performance for the top feature extraction methods (bigram binary and HMM methods), two types of comparison graphs are presented: A chart for the family-wise F-measure values against these two methods is given in Figure 4.3, while Figure 4.4 shows the confusion matrices for classification using the two methods respectively.

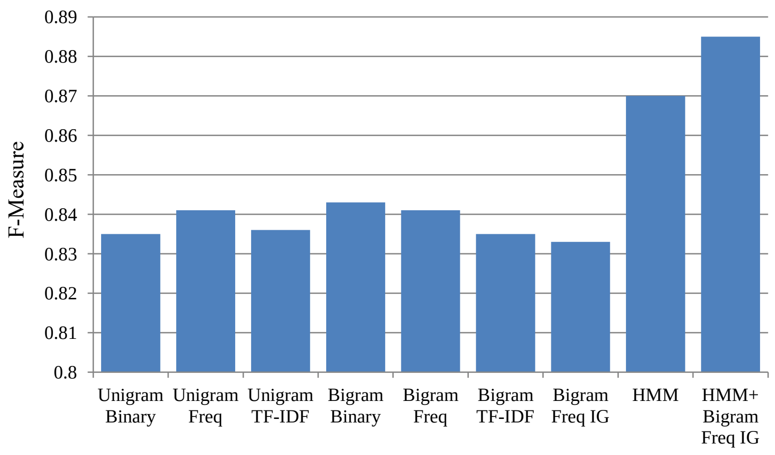


Figure 4.2: Comparison of feature extraction methods using Random Forest classifier

As it can be observed in Figure 4.2, features extracted through Hidden Markov Model were the best representatives of malware behavior among all the other individual methods. Classification using HMM features resulted in the highest F-measure value of 0.87, which was 3.2% more than the next highest F-measure of 0.843 for the bigram binary features. This result, combined with the observations made from Figure 4.1, signifies the expressiveness of HMM based features since they were responsible for best classification while being smallest in number among all other feature extraction methods. It also shows that HMM performs well as a feature based classification method in comparison with other conventional sequence classification methods for classification of malware.

A comparison of family-wise F-measures for bigram binary and HMM features is given in Figure 4.3. Although for most of the malware families the classification results for features extracted through the two methods are close, significant difference is observed in a few malware families such as BIFROSE, BUZUS, FRAUDLOAD and POISON. For these malware families, the bigram binary features had scored a relatively low F-measure, but the HMM based features proved to be more discriminative as depicted by their higher F-measure scores.

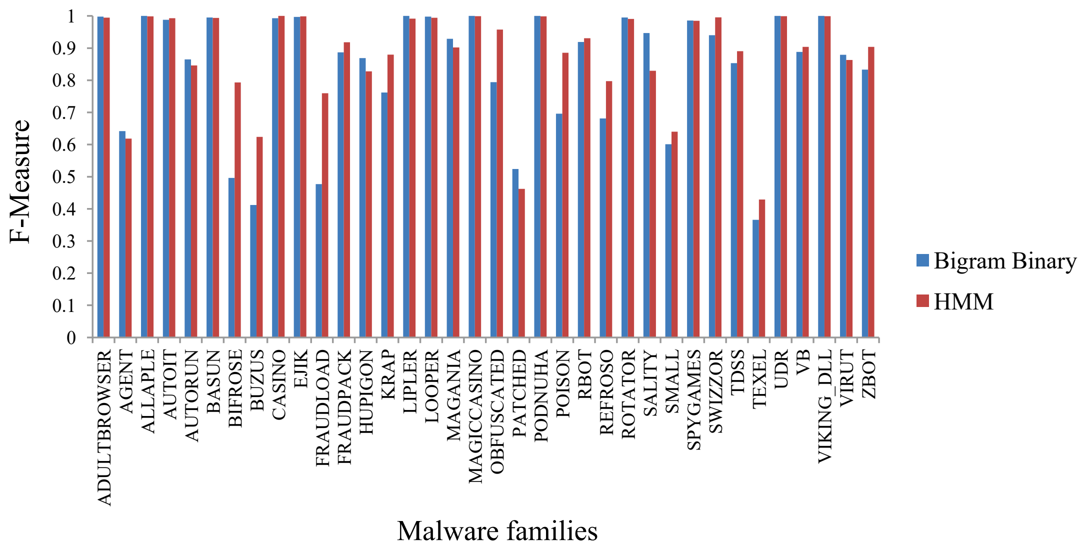
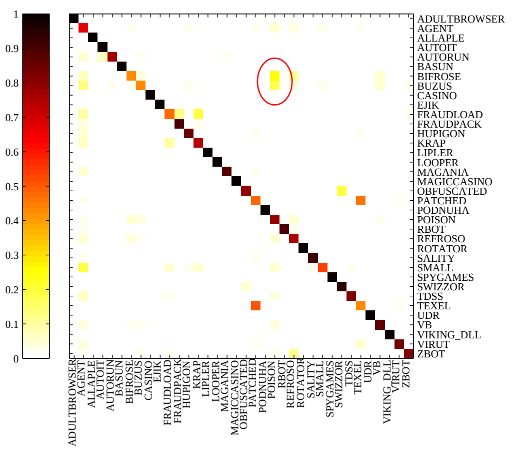


Figure 4.3: Family-wise F-measure for bigram binary and HMM methods

Figures 4.4(a) and 4.4(b) show the confusion matrices for classification using the bigram binary and HMM-based feature extraction methods respectively. The vertical axes in these confusion matrices represent the actual malware families, whereas the horizontal axes depict the predicted families. The shade of the box at the intersection of i^{th} row and j^{th} column of the matrix represents the ratio of samples of i^{th} malware family classified as j^{th} family to the total number of i^{th} family samples. The sidebars show the shades against ratio values: the darker the shade, the greater the ratio. Consequently, the darker shades along the diagonal


(a) Using bigram binary features

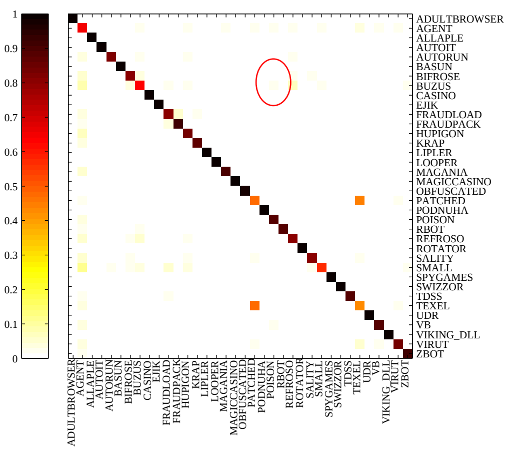

(b) Using HMM features

Figure 4.4: Confusion matrices for classification

show that most of the samples were correctly classified. On the other hand, dark colored boxes on either side of the diagonal represent misclassified samples.

The confusion matrices in Figure 4.4 reveal the reason for the difference seen in F-measures for bigram binary and HMM-based features. The confusion matrices are mostly similar, hinting that the classifier was able to discriminate between malware families on the basis of features extracted through the HMM and bigram binary methods alike. The difference is especially visible for BIFROSE and BUZUS samples (shown in oval) which were misclassified as belonging to POISON family in case of bigram binary features, but for HMM based features these misclassifications were significantly less. This is an indication of supremacy of HMM based features because the classifier was able to differentiate between closely resembling families (BIFROSE and POISON, for example) on the basis of these features.

The feature extraction methods using bigrams were only marginally better than unigram based methods with the exception of binary feature method, in which case the unigram method showed slight improvement over bigram features. At first glance this observation may seem to be erroneous, since losing the sequence information in case of unigram should result in higher false positive rate, thus making significance difference in the overall classification performance. A plausible explanation for comparable results is that the system calls for performing a particular task are usually grouped together, and the sequence of calls within a short group is generally maintained across all malware families. Therefore, there do not seem to be many out of sequence system calls present in the dataset which would lead to false negatives. One could, however, engineer a sequence by injecting dummy or redundant system calls thus making a new malware look different from the known malware, which would result in a misclassification.

Applying TF-IDF weighting scheme had an insignificant, albeit slightly negative, effect on classification performance as compared to using absolute (raw) frequencies on both unigram and bigram sets. This result suggests that the patterns of individual system call frequencies are quite specific to malware families, therefore using the normalized and corpus-wise frequencies did not add any useful and discriminative information to the feature set.

Binary feature method remained below frequency feature scheme in unigram based representation but was better using bigrams, although the difference was negligible in both cases. The fact that binary and frequency methods produced similar results further strengthens the earlier observation that system calls made by malware belonging to a specific malware family are distinct from other malware.

The average F-measure for the bigram frequency IG feature set, obtained after applying feature selection to the bigram frequency set, decreased to 0.833 from 0.841 for the original feature set. In other words, an approximately 1 decrease in classification performance was observed for a 96.4 decrease in the number of features. The loss in classification performance is negligible against the reduction in features, yet these two factors should be weighed before using feature selection in a given malware analysis scenario.

In case of the combined HMM and bigram frequency IG feature vector, the average F-measure increased to 0.885, which is a gain of 1.7% as compared to the F-measure value of 0.87 for the individual HMM feature set, and 5.2% more against the F-measure of 0.841 for bigram frequency IG feature set. This is an encouraging result and paves the way for further research on using various combinations of feature vectors for the task of malware detection and classification.

Chapter 5

Effect of number of hidden states on classification performance using HMM

Extensive research is being carried out in the domain of malware analysis and detection to counter the rapidly growing malware threat. Among all these malware detection methods, statistical and probabilistic schemes have been shown to be more effective than others. As a consequence, more and more research is being conducted on use of machine learning techniques, which have their roots in statistics and probability, for analyzing and detecting malware. Recent research in the domain of malware analysis has seen growth in use of Hidden Markov Model (HMM) for tasks such as malware classification and clustering. Researchers have proposed various ways of exploiting the powerful pattern matching capabilities of HMM for differentiating malware from cleanware, as well as for classifying among malware families. In this chapter the current state-of-the-art in HMM-based malware analysis is extended in two dimensions. First the impact of number of hidden states on classification performance of the proposed HMM-based malware classification technique is studied. Led by the observations from this experiment, the HMM component of the classification method is replaced with a Markov Chain Model (MCM), and the proposed method is compared with the HMM-based method from effectiveness and efficiency perspectives.

HMM has been discussed in detail in Chapter 2; to fix ideas a brief overview is provided here. An HMM is used to model a given sequence as a doubly stochastic process. If treated as a state machine, HMM represents a process in terms of *hidden* states through which the process transits according to a probability distribution often represented as a state transition matrix (referred to as A matrix in literature). Progress of the process (the state machine) is observable through symbols which are emitted by each state following a second probability distribution called the

B matrix. Given a sequence, the Baum-Welch algorithm (Rabiner, 1989) is used to estimate the optimal A and B matrices that best fit the given observations (elements of sequence) into the specified number of hidden states.

The number of hidden states, thus, plays a vital role in the HMM modeling process. Unfortunately, there is no fixed rule for determining a globally optimal value for this parameter; an understanding of the data being modeled is necessary for specifying the number of states in a given problem. An interesting example concerning application of HMM for modeling text can be found in (Cave and Neuwirth, 1980). The experiment involved modeling a two-state HMM over characters extracted from a well-known piece of English literary text. After the HMM had been trained, analysis of B matrix revealed that the two states corresponded to vowels and consonants of the English language. It can be inferred from this experiment that HMM states used for modeling the given data should represent some intrinsic structural unit in the data, and this factor should be reflected in the specified number of hidden states. Similarly, while using HMM in speech recognition applications, states could represent the phonemes in case of discrete word analysis (Rabiner, 1989), or acoustic-phonetic units of the language in case of continuous speech synthesis (Levinson, 1987).

A survey of literature on application of HMM in malware analysis shows that the number of HMM states for modeling malware behavior has been usually decided using the trial and error approach by experimenting with different values and selecting the one which produced the best results. In other words, concrete reasoning behind using a particular number of hidden states, or an analysis of the underlying properties of data, has not been generally demonstrated by researchers. Austin et al. (2013) went a step further and studied the transition and observation probability matrices against different number of states to find the significance of these probabilities. They reported their observations on the groupings of symbols within the hidden states which, although being valid information, may not be helpful in determining an optimal number of hidden states for their data. Furthermore, there has not been any attempt to study the impact of the number of hidden states on performance of any given HMM-based classification method in terms of classification accuracy or efficiency.

In this chapter first the effect of number of HMM states on classification performance is analyzed using the similarity-based malware classification method on two different datasets containing behavioral reports of real malware in terms of system call and API call logs. The number of HMM states is varied according to *justifiable* criteria, and the outcome is analyzed with an aim to find a relationship between the number of states and classification performance. Observing that the best performance is obtained in case of the number of states being equal to the number of unique observation symbols in the data, it is proposed to model the malware behaviors using MCM in place of HMM and the classification performance as well as efficiency aspects of the two approaches are compared on a larger and more diverse system call dataset.

This chapter proceeds as follows. The next section describes the experiments for observing the effect of number of hidden HMM states on performance of malware classification and discusses the outcome. Section 5.2 gives a brief overview of Markov Chain Model and discusses time complexities of training and evaluating Markov Chain Model and Hidden Markov Model. Section 5.3 reviews some previously proposed methods for malware classification involving MCM. Comparison of MCM and HMM is provided in Section 5.4, and results of the comparison are discussed in Section 5.5.

5.1 Studying the effect of HMM states on classification performance

For conducting this study the similarity-based malware classification scheme presented in Chapter 3 was adopted. The proposed technique was used to observe the impact of number of hidden states on malware classification using two different datasets consisting of system call and API call logs of real malware samples. For reference, a brief description of the malware classification method is provided here which is followed by details of experiments using both the datasets.

Table 5.1: System call dataset for study of hidden states

S. No.	Malware Family	Number of samples	Observation sequence length			
			Min.	Max.	Avg.	Combined
1	ADULTBROWSER	262	717	1011	739	193,618
2	AGENT	400	16	89,031	2931	1,172,304
3	BUZUS	142	15	51,697	2819	400,298
4	CASINO	140	296	1900	409	57,260
5	EJIK	168	236	250	242	40,656
6	TEXEL	400	15	43,528	316	126,485
	Total:	1512				1,990,621

5.1.1 Malware classification using HMM

The proposed malware classification scheme consists of two steps. In the first step, malware families are modeled using HMM and then each malware sample is evaluated against each of the HMMs in order to obtain its similarity score in the form of a feature vector. In the second step, feature vectors for all the malware samples are used to train a distinctive classifier which can then classify a malware sample, given its similarity vector, into one of the malware families.

5.1.2 Using system call dataset

The first dataset that was used for this study is a subset of data employed in the earlier experiment (described in Chapter 3) which, in turn, was reduced and transformed from (Rieck et al., 2011). The dataset includes system call logs of malware recorded by executing them in CWSandbox (Willems et al., 2007), and converted to numerical representation as described in Chapter 3. Based on observations from the cited work, six malware families were selected including three families for which the HMM-based malware classification method had performed well and three families for which the classification results were below average. Table 5.1 provides details about the malware families including the number of samples as well as the minimum, maximum, average and cumulated number of observations in each family.

For the current experiment, a total of six HMMs were trained, each representing behavior of a particular malware family. Evaluation of malware samples produced a set of feature vectors including 1,512 instances, each representing a malware sample's similarity with all the malware family models.

This experiment was performed in five stages, modeling the HMM with a different number of states (2, 5, 10, 20 and 120) in each stage. The number of states in the lower range (2 and 5) were chosen because earlier research (Wong and Stamp, 2006) had shown promising results with these numbers of states. On the other extreme 120 was the biggest number of states to experiment with, which corresponded to the total number of unique system calls in the data, as suggested in (Warrender et al., 1999). Similarly, the number 20 represented the number of system call categories. The category information is not explicitly present in the data, and the reason for experimenting with this value was to see if HMM trained with 20 states can somehow reflect this implicit feature of data. In between the extremes, data was modeled using 10 states as a median value. Figure 5.1(a) plots the F-measure of HMM-based classification method against different number of states.

5.1.3 Using API call dataset

In order to ascertain the findings from the first experiment using system call dataset, the same experiment was performed on another dataset which comprised of API calls recorded through dynamic analysis of malware samples. API calls are another effective representation of a program's actions, and therefore considerable research in malware analysis is based on such calls (Shankarapani et al., 2011; Alazab et al., 2011; Elhadi et al., 2014). There are two ways of extracting API call sequences corresponding to a program: static and dynamic. In static method, API calls are extracted from the disassembled binary code of the program file, while dynamic API calls are recorded by executing the program in a sandbox or virtual machine. Since dynamic API trace captures the actual actions that the program performed during execution therefore the dynamic API calls were used in this experiment.

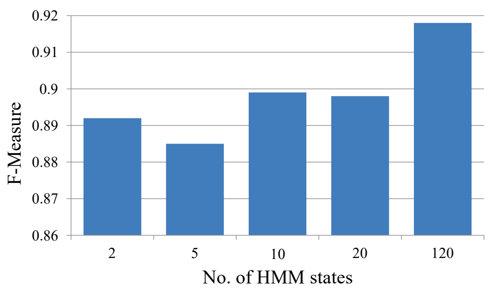
Table 5.2: API call dataset for study of hidden states

S. No.	Malware Family	Number of samples	Observation sequence length			
			Min.	Max.	Avg.	Combined
1	AGENT2	244	1	26,980	1359	331,596
2	BIFROSE	225	2	30,841	1753	394,425
3	DELF	230	1	148,490	6784	1,560,320
4	POISON	248	1	80,192	2395	593,960
5	VB	250	85	139,530	5879	1,469,750
	Total:	1197				4,350,051

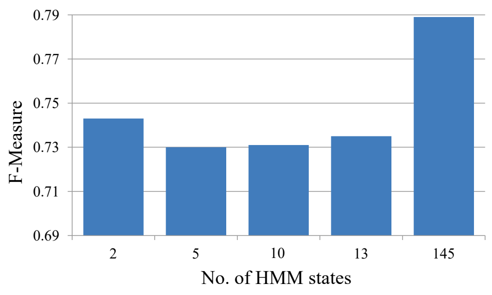
To prepare the API call dataset first 1,197 labeled malware samples belonging to five malware families were downloaded from VX Heaven[1] which is an online malware repository. The malware samples were then analyzed using an online dynamic analysis service, malwr[2], which generated analysis reports for all samples. A typical report includes static as well as dynamic analysis but since only the API call trace was required therefore the sequences of API calls were extracted from all the reports. A total of 145 unique API calls were identified from the API sequences. The next task was to replace the system call names with unique identifiers so the API calls sequences were represented as sequences of arbitrary lengths, containing numbers in the range from 1 to 145. Table 5.2 provides details about malware families included in the API call dataset.

The number of hidden states for this experiment on API call dataset were selected to be 2, 5, 10, 13 and 145, where the values 13 and 145 corresponded to the number of API call categories and unique API calls present in the data, respectively. Following the same methodology as that used for the system call data, first a total of 1,197 feature vectors were generated using the HMM modeling and evaluation in Matlab, followed by classification of these feature vectors using the Random Forest classifier in WEKA. Classification results against various number of hidden states for the API call dataset are shown in Figure 5.1(b).

[1]`http://vxheaven.org/`
[2]`https://malwr.com/`



(a) System call dataset



(b) API call dataset

Figure 5.1: Impact of number of HMM states on F-measure

5.1.4 Observations from the study

The graphs of Figure 5.1 show that for both the datasets no linear relationship exists between the number of hidden states and classification performance. However, one common finding from the two experiments is that maximum F-measure is achieved when states are kept equal to the unique observation symbols in the data. From this observation it is hypothesized that the simple Markov Chain Model (MCM) should be a better choice of model in this scenario since it also treats each symbol as a separate state and, therefore, should be at least as effective as HMM. Furthermore, modeling an MCM is theoretically more efficient than HMM as discussed earlier. To test this hypothesis experiments were performed on larger dataset consisting of system call logs of real malware using the same malware classification method, once modeling the malware behavior using HMM and then with MCM. These experiments are discussed next.

5.2 Markov Chain Model

Here a review of the Markov Chain Model is provided. The theoretical time complexities for training and testing phases of MCM and HMM are also discussed.

A Markov chain is a probabilistic process, often represented as a finite state machine, which undergoes state change according to a set of state transition probabilities denoted by a state transition matrix. The underlying assumption behind a Markov chain is that the process obeys the Markov property, that is, every state is only dependent on previous k states for a k^{th}-order Markov chain. The simplest form of Markov chain is the first-order chain in which the next state can be predicted on the basis of current state only.

A Markov Chain Model can be used to represent an arbitrarily long sequence whose elements come from a finite set. In such a model, the sequence is assumed to be a Markov process such that the states come from a finite state space (usually composed of unique items present in the sequence) while two consecutive items of the sequence represent a state transition. A given Markov chain $x = \{x_1, x_2, x_3, ..., x_q\}$ can be modeled by a state transition matrix A, which stores the transition probabilities for all pairs of states i and j as given by Equation 5.1.

$$A_{i,j} = P(x_t = j | x_{t-1} = i) \tag{5.1}$$

To calculate these probabilities, the Maximum Likelihood Estimation procedure can be adopted using the counts, for all pairs of states i and j, of the number of transitions from state i to state j and the total number of transitions from state i, as represented by Equation 5.2.

$$A_{i,j} = \frac{N_{i,j}}{\sum_{k=1}^{n} N_{i,k}} \tag{5.2}$$

Elements of the transition probability matrix are row-stochastic, such that:

$$\sum_{j=1}^{n} A_{i,j} = 1, i = 1, 2, 3, ..., n \tag{5.3}$$

where n represents the total number of states in the model. It may be noted that this method of learning the Markov Chain Model is also applicable for modeling a set of sequences representing processes that share some common properties (such as behavior reports of different malware of the same family). In such a case, calculation of $N_{i,j}$ and $N_{i,k}$ in Equation 5.2 is performed over all the sequences.

Once a model has been derived for a given sequence (or a set of sequences) in the form of a state transition matrix, the probability that another sequence

$y = \{y_1, y_2, y_3, ..., y_r\}$ is generated by the same model can be calculated by using Equation 5.4 (Durbin et al., 1998).

$$P(y) = P(y_r|y_{r-1})P(y_{r-1}|y_{r-2})...P(y_2|y_1)P(y_1) \tag{5.4}$$

Simplifying Equation 5.4 yields:

$$P(y) = P(y_1)\prod_{i=2}^{r} P(y_i|y_{i-1}) \tag{5.5}$$

The transition probabilities can be replaced by values in the transition probability matrix, resulting in Equation 5.6.

$$P(y) = P(y_1)\prod_{i=2}^{r} A_{y_{i-1},y_i} \tag{5.6}$$

Since computing the product of probabilities can lead to underflow therefore it is convenient to use the log of probabilities and sum them up instead. Also, the value $P(y_1)$ represents the probability of the sequence to start from the first symbol, and is generally ignored for simplicity, further reducing the Equation 5.6 as:

$$P(y) = \sum_{i=2}^{r} log(A_{y_{i-1},y_i}) \tag{5.7}$$

5.2.1 Time complexity of Markov Chain Model

Time required to compute the transition matrix of a given sequence (also referred to as the time for modeling the sequence later in this document) containing n elements is $\mathcal{O}(n)$, because just one pass over the sequence is needed. Similarly, computing the probability of a given sequence against a model (state transition matrix) is also linear with respect to the number of elements in the sequence.

5.2.2 Time complexity of Hidden Markov Model

Time complexity of the forward algorithm, which is a key part of both the Baum-Welch and forward-backward algorithms, is reported as $\mathcal{O}(N^2T)$ in the literature

(Rabiner, 1989), where N represents the number of hidden states and T is length of the sequence. It may be noted that Baum-Welch is an iterative algorithm which continues for a predefined number of iterations or unless the estimated parameters in two consecutive iterations are within a specified threshold. Therefore, the time complexity for the Baum-Welch algorithm gets multiplied by the number of iterations. Similarly, the forward-backward algorithm for computing similarity score of a sequence against a given model also uses a backward algorithm in addition to the forward algorithm (hence the name forward-backward) therefore its time complexity becomes $\mathcal{O}(2 \times N^2T)$ in reality. Although multiplication with a constant term is ignored while computing the theoretical time complexity of algorithms, these multiplicative factors do play a significant role in real applications, as will be shown and discussed in Section 5.5.

5.3 Markov Chain Model for malware classification

In this section a few representative research efforts in the field of malware detection and classification are discussed that are based on Markov Chain Model.

In one of the previously proposed malware detection and classification schemes based on Markov chains, Shafiq et al. (2008) used the byte representations of different kinds of documents as 1st order Markov chains in which the 256 states corresponded to all possible byte values. It was shown that malware detection in documents was possible by using the difference in entropy rate of the state transition matrices for original (benign) and infected version of the same file.

Andserson et al. (2011) generated Markov graphs from instruction traces of programs, and used a combination of Gaussian and spectral kernels as the similarity measure in Support Vector Machine for malware detection as well as classification. The authors reported that the proposed technique performed better than malware detection methods based on n-gram and signatures.

In (Colbaugh et al., 2013), Colbaugh et al. represented opcode (operational code) sequences of benign and malware programs as Markov chains, and used Kullback-Leibler divergence to find differences between state transition matrices computed from these Markov chains. The pair-wise differences of each program with every other program were concatenated in form of feature vectors, which were then used for malware detection by performing a binary classification with the help of a semi-supervised classifier.

In a similar approach, Wang et al. (2013) calculated pair-wise similarity between two malware samples by computing the probability of each sample against the state transition matrix of the other, and used the sum of these probabilities as the distance measure for feature vector generation. Clustering of malware on the basis of generated feature vectors showed encouraging results.

Storlie et al. (2014) have proposed and implemented a system for malware classification. The system models dynamic instruction trace as Markov chain and creates a probability matrix based on transitions between instructions. Using the logistic spline regression algorithm on the probability matrices, the system performs statistical classification on dynamic traces of malware and benign programs. To filter the insignificant model parameters from the high number of actual parameters, the authors have adopted the Relaxed Adaptive Elastic Net estimation algorithm. The resulting system is able to perform classification between malicious and benign programs with an accuracy of 97.6%. The authors conclude that other model-based approaches based on Markov chains may also prove to be effective for malware classification.

5.4 Comparison of Hidden Markov Model and Markov Chain Model

To evaluate and compare HMM and MCM the same system call dataset (Table 3.1) was chosen as used in experiments discussed in Chapter 3 with the exception of BUZUS and ROTATOR families. The reason for omitting these two families from the current experiment was that the number of observation symbols in these two families was very large (around 30 million combined) and modeling these families with 120 states was taking a long time. It may be noted that even without including these two malware families, the size of dataset is comparable to average dataset size used in experiments previously reported in literature (Table 2.2).

5.4.1 Malware classification using HMM

The two differences between the first study on hidden states and this experiment lie in the size of dataset and number of hidden states used. In this experiment the number of malware samples was 8,128 therefore the feature space generated by the HMM modeling and evaluation stage included 8,128 feature vectors, each with 34 features corresponding to the similarity scores of malware samples against the 34 modeled HMMs. Furthermore, HMM modeling was done with 120 hidden states. As before, the feature vectors were classified using Random Forest classifier in WEKA.

5.4.2 Malware classification using MCM

For performing malware classification using the Markov Chain Model, the first step of the aforementioned HMM-based malware classification method was modified by modeling the malware families using Equation 5.1, and evaluating all malware samples against all models using Equation 5.7. The equations for sequence modeling and evaluating were implemented in Matlab R2013a. The obtained feature vectors were subjected to the classification process using Random Forest classifier in WEKA. 10-fold cross validation was performed for evaluating the classification performance.

5.4.3 Efficiency aspects

In order to perform a comparative analysis of HMM and MCM methods from efficiency angle, the sequences representing system call logs of VIKING_DLL family were taken and ten sets of sequences were extracted such that the combined number of system calls in each set was multiple of 30,000. The idea was to evaluate the scalability of these two methods by observing time requirements against various sequence lengths. The HMM- and MCM-based training and testing procedures were performed for the ten sets, and the Matlab code for modeling and evaluating the two methods were enclosed within pairs of `tic` and `toc` commands to record the time consumed by these tasks for each set. For HMM modeling the number of hidden states was set to 120. Since the classification time in WEKA is the same for both the methods therefore this factor was excluded from time complexity comparison.

The experiments were performed on a system with 2.5 GHz Intel® Core™ i5 processor and 4 GB RAM.

5.5 Results and discussion

Figure 5.2 shows the comparison of HMM- and MCM-based classification methods for the system call dataset in terms of F-measure. For certain malware families such as ALLAPLE, EJIK, CASINO etc., both the methods obtained comparable, high F-measure scores. For most of the malware families, though, F-measure scores for the two methods were different; in some cases the HMM-based method performed better while MCM-based method was winner in others. Overall performance of the MCM-based classification method in terms of weighted average of F-measure score over all the malware families was the same as the HMM method.

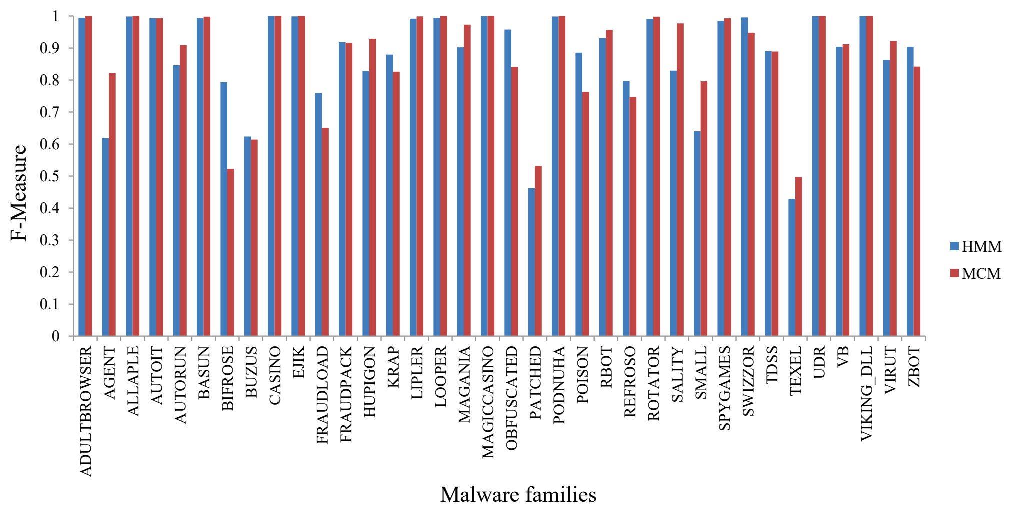
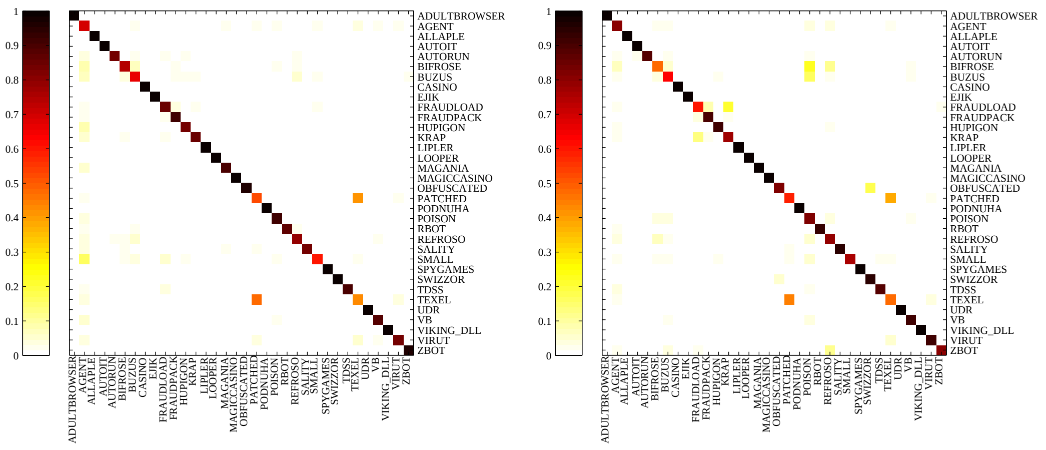


Figure 5.2: Comparison of F-measures for HMM- and MCM- based classification



(a) HMM (b) MCM

Figure 5.3: Confusion matrices for classification

It is obvious from Figure 5.2 that both these methods performed differently for most of the malware families, and therefore no clear advantage of either method can be ascertained over the other. Apart from the case of TEXEL and PATCHED family malware, where both the methods misclassified in a similar fashion, the confusion matrices for these methods show gray boxes in different regions as depicted in Figure 5.3. From this result it was intuited that a combination of HMM and MCM might improve the results of the individual modeling methods. To evaluate this idea the feature vectors generated by both the models were concatenated so that each feature vector represented a malware sample's similarity with 34 HMMs and 34 MCMs, resulting in 68 features. Classifying the feature vectors using Random Forest classifier in WEKA confirmed the intuition, and the combined HMM and MCM methods gained almost 1.5% in F-measure over the individual methods as shown in Table 5.3.

Figure 5.4 plots the training (modeling) and testing (evaluation) times required by the HMM and MCM methods against sequences of different lengths. It may be noted that a comparison between the HMM and MCM modeling methods

Table 5.3: Classification results in terms of weighted average F-measure

F-measure			Gain
HMM	MCM	Combined	
0.869	0.87	0.883	1.49%

could not be presented using a graph in linear scale due to the huge difference in execution time for these two methods, and therefore logarithmic scale has been used. Slopes of all the curves in the figure may look similar but in fact the HMM curves are much steeper than MCM curves, signifying that modeling time in case of HMM increases much faster than its MCM counterpart as the length of sequence increases. It is further emphasized that the comparison shown in Figure 5.4 was made on sequence lengths of 30,000 to 300,000 whereas the cumulative length of all the sequences in the dataset exceeds 35 million. Hence for evaluation of the two methods on the whole dataset, it took several days for modeling all the malware families in the dataset using HMM while the MCM learning was completed in a few seconds.

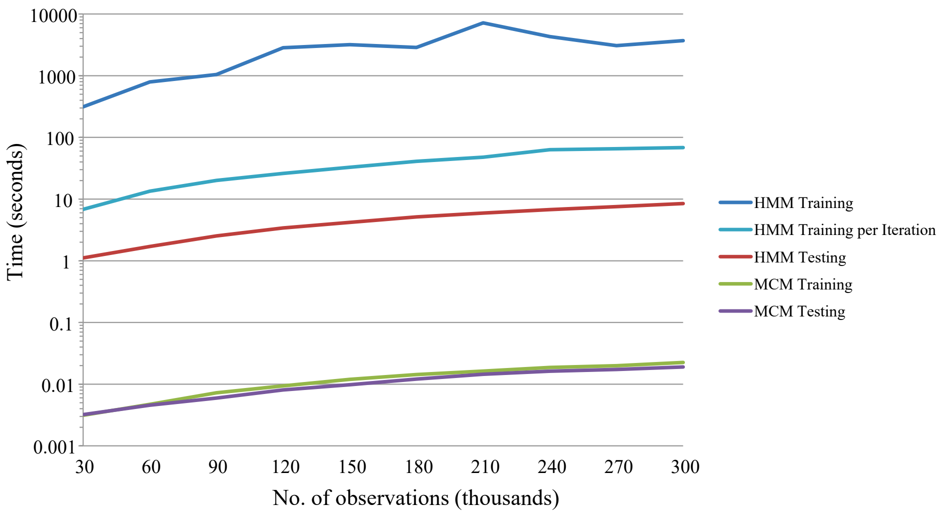


Figure 5.4: Time required for training and testing the HMM and MCM

An interesting observation that can be made from Figure 5.4 is that HMM modeling time does not always increase with the number of observations in the training sequence. This is because one important aspect of HMM modeling, which is not shown in this graph, is the number of iterations took by the HMM modeling algorithm to optimally estimate HMM parameters. When learning the parameters of an HMM model, i.e. the three probability distribution matrices, these matrices are initialized with random values. Therefore HMM modeling itself becomes a non-deterministic process such that the number of iterations for obtaining an optimal model depends upon the initial probability distributions. That is why the curve for HMM modeling time shows dips and spikes. To give an idea of the relationship between modeling time and length of the sequence, Figure 5.4 also plots the HMM modeling time *per iteration* against the sequence length. It is clear from this graph that as the number of observations in the sequences grows, the HMM modeling time grows steeply and therefore HMM may not be suitable for any practical application involving large volumes of sequential data pertaining to malware behavior.

Figure 5.5 shows the number of iterations against number of hidden HMM states for various sequence lengths. An upward trend is observable in the graph from left to right, signifying the general rule that for any given sequence length the number of iterations required to reach satisfactory model parameters increases with the number of hidden states. That is the cause of increased time requirements with higher number of hidden states when training an HMM, and therefore training HMMs with the number of states equal to the number of observations is not practically viable.

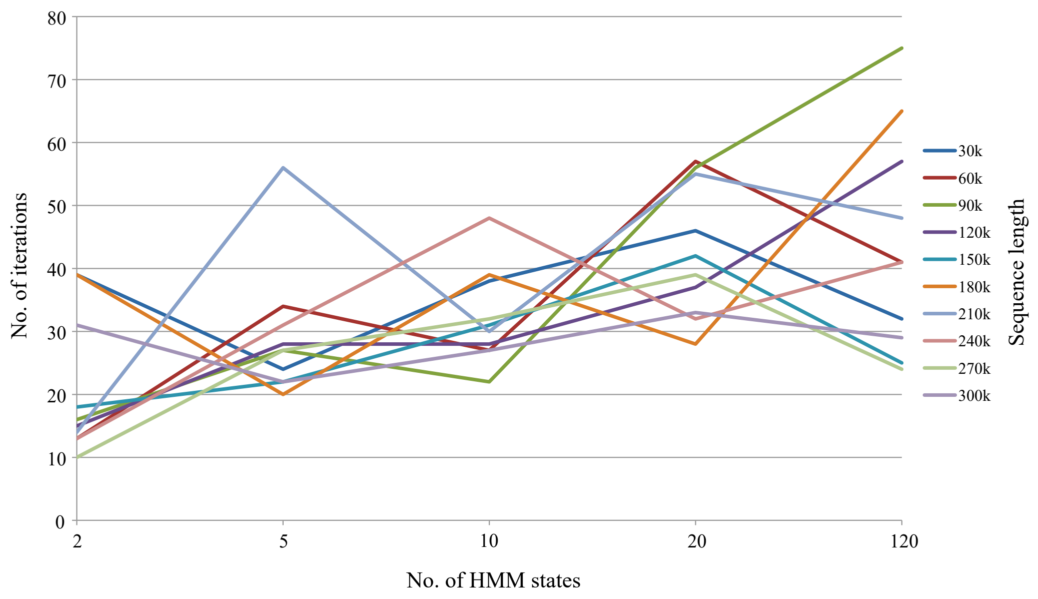


Figure 5.5: Number of iterations against sequence length for HMM training

Whereas the time required for modeling a sequence using the Markov Chain Model depends only on the number of items in the sequence, learning the parameters of the same sequence for a Hidden Markov Model takes into account two more factors: the number of hidden states and the number of iterations. These additional factors heavily effect the computational time required for modeling and evaluating sequences. In addition, the optimal number of states for the malware classification scenario depends on the underlying data and there does not exist a fixed formula for determining this important parameter for HMM modeling. It, therefore, becomes a matter of trial and error to find the optimal number of states for modeling malware behavior. This fact further weakens the case of using HMMs for a realistic application in malware analysis domain.

Researchers have tried to model malware behaviors using HMM with different, usually randomly chosen, number of states with a hope to capture some hidden pattern inside the data. This dissertation argues that HMM may be very effective for sequential pattern matching and classification tasks in some domains, but its usefulness for malware analysis cannot be ascertained unless concrete reasons are established for specifying the state configuration for some given data. It can thus be concluded that if HMM needs to be applied to the malware classification problem in a meaningful way, a deep understanding of the data to be modeled is required so that HMM parameters can be set to justifiable rather than random values.

The results of comparison conducted in this chapter confirm the hypothesis that Markov Chain Model performs at least as well as the Hidden Markov Model in a malware classification application. Furthermore, there is a clear indication that the MCM-based malware classification scheme is superior to the HMM-based method from the efficiency standpoint. Although the advantages of using Markov chain method over Hidden Markov Model have been shown using just one malware classification scheme, it is postulated that other malware analysis methods which are based on HMM can also benefit from the observations made in this research.

Chapter 6

Conclusions

With an ever increasing number of computer devices, including but not limited to personal computers, servers and mobile phones etc., the playground for malware developers has becoming vast. Availability of malware creation kits and obfuscation tools has made the task of cyber security professionals hard since new malware variants can be quickly flooded into the computer networks with little effort.

The huge number of new malware variants reported everyday have to be analyzed in order to determine the family of known malware they belong to. This information is necessary for the developers of ant-malware products and services for an efficient update and delivery of the respective signature to the client computers worldwide. Malware classification has thus become an active area of research in the broader domain of cyber security.

After a careful analysis of the prevailing situation of malware threat and critical review of literature, this dissertation identified that effective and efficient malware classification is an open research challenge. The particular focus of research for this dissertation was the evaluation of a popular and effective sequence classification tool, Hidden Markov Model, in combination with dynamic program features, for malware classification problem. Hidden Markov Models have been widely and effectively used for sequence classification problems in domains sch as speech analysis, behavior modeling and handwriting recognition etc., and therefore their use for malware classification was also in order.

The problem statement for this thesis raised three research questions:

1. How can HMM be used to classify malware on the basis of their behavior?
2. What is the role of number of hidden states when using HMM for malware classification?

3. How efficient is HMM based malware classification to be used in practical situations?

Chapter 3 of this dissertation addressed the first research question by proposing a method of using Hidden Markov Model to classify malware on the basis of a sequential representation of malware behavior, and analyzing the effectiveness of the proposed method on behavioral profiles of real malware in terms of system calls. The proposed method was compared against other methods including the state-of-the-art, and it was concluded that HMM can be used for effectively classifying unknown malware samples into known malware families on the basis of malware's dynamic behavior.

The similarity-based method, proposed in this research, can be regarded as a feature based sequence classification technique, since it transforms variable length behavioral profiles into fixed width vectors which can be used to train a discriminative classifier. Evaluating the effectiveness of similarity-based malware classification method with respect to this aspect was an extension of the first research question, and was discussed in Chapter 4. After a survey of literature on feature extraction methods used for converting variable length sequences into fixed length feature vectors in malware analysis domain, various commonly used methods were implemented and compared against the HMM-based method by using the obtained feature vectors to train a discriminative classifier. The results of this experiment revealed that the HMM-extracted features had more representative power than other methods because they resulted in better classification performance with fewer number of features, and hence HMM's significance as a feature based malware classification method was ascertained. It was also learned through experimentation that a combination of HMM-extracted features with other features can enhance the effectiveness of feature set.

The second and third research questions were answered in Chapter 5, which focused on studying the impact of number of hidden states on the classification accuracy of the proposed similarity-based malware classification method. This study yielded interesting results about the suitability of HMM for malware classification task. Based on observations from the study, an alternative method of malware classification was proposed which made use of the simpler Markov Chain Model (MCM). Upon comparison with HMM based method, the MCM based malware classification technique proved to be much more efficient without compromising the desired accuracy. The study also concluded that in case HMM performs well in a given scenario, its results can be further improved by using it in combination with MCM at negligible extra computational cost. One important observation from this study is that Hidden Markov Model has been used for the task of malware detection and classification without a deeper understanding of the concept, and that the properties of data to be modeled are not being thoroughly analyzed. This leads to a situation where the learned model does not relate to the hidden semantics of the data, and therefore defeats the very purpose behind using the Hidden Markov Model.

In summary, this research filled a significant gap in the HMM-based malware classification domain by identifying and addressing shortcomings in the previously proposed schemes found in literature. The contributions of this research are:

1. A broad literature survey was conducted covering the classic as well as the state-of-the-art approaches in malware analysis, detection and classification.
2. Previously proposed HMM-based malware classification methods were critically analyzed and evaluated against criteria inferred from the literature.
3. A novel method of malware classification were proposed which involved training of Hidden Markov Models over behavioral profiles of malware.
4. The proposed method was evaluated using comprehensive dataset containing behavioral reports of real malware
5. Another comparison was performed between the proposed similarity-based method and various other feature extraction methods to further strengthen the effectiveness of proposed method.

6. The effect of number of hidden HMM states on classification performance was studied and consequently the HMM's efficiency aspects were analyzed.

7. Identifying the HMM modeling and evaluation stages as the performance bottlenecks for the proposed malware classification scheme, an enhancement was proposed and evaluated that replaced HMM with Markov Chain Model.

8. A novel combination of HMM and MCM based methods was proposed and evaluated for the task of malware classification.

A comparison of the proposed similarity-based method with previous HMM-based malware classification approaches, in light of the criteria proposed in Chapter 2, is presented in Table 6.1.

Table 6.1: Comparison of HMM-based malware classification methods

S.No.	**Reference**	**Feature**	**Performance (metric)**	**Dataset size**	**Comparison**	**States analysis**	**Efficiency analysis**
1	(Warrender et al., 1999)	Dynamic (Sys. calls)	96.9% (TPR)	92,000	✓		
2	(Wong & Stamp, 2006)	Static (opcode)	Not reported	200			
3	(Attaluri et al., 2009)	Static (opcode)	˜67% (Det. Rate)	280			
4	(Ravi et al., 2013)	Dynamic (Sys. calls)	0.964 (Acc.)	19,000			
5	(Austin et al., 2013)	Static (opcode)	87% (Det. Rate)	60		✓	
6	(Annachhatre et al., 2014)	Static (opcode)	0.94 (AU-ROC)	8,119			
7	(Damodaran et al., 2015)	Static, dynamic, hybrid	N/A	785			
8	Proposed approach	Dynamic (Sys. calls)	0.994 (F-measure)	8,182	✓	✓	✓

6.1 Limitations

The malware classification method proposed and evaluated in this research involved training probabilistic models on malware's dynamic behavioral features. As the literature suggests, dynamic analysis has certain limitations such as time required for executing the malware samples and recording their behavior, inability to cover all execution paths and random behaviors, etc. The work presented herein is thus prone to the same limitations.

6.2 Future work

Some possible directions for future research on the topic addressed in this dissertation are described below:

- This work can be extended in the future by using other presentations of malware behavior such as opcodes or instruction sequences for evaluating the HMM-based malware classification methods proposed in this dissertation. Similarly, experimenting with hybrid set of features might also be an interesting extension of the presented work.
- The feature extraction methods used in this research (discussed in Chapter 5) have been evaluated on system calls at a somewhat middle level of granularity. A higher level, according to MIST format, would be the category level (file system, registry, communication, etc.) and a lower granularity would include system call parameters as well. Experimenting with lower granularity may produce better results for all the schemes but will incur more computational overheads since the combination of a system call with different parameters will result in a greater number of distinct elements in the term dictionary. A possible extension of this study, thus, could include evaluating the feature extraction methods for different granularities of the input sequences.
- The effect of number of hidden states has been observed from the performance perspective in this research. Analysis could also be performed from

another angle by studying the A and B matrices for different numbers of hidden states in order to get an insight into what the hidden states represent for malware classification task.

References

Agrawal, H., Bahler, L., Micallef, J., Snyder, S., and Virodov, A. (2012). Detection of global, metamorphic malware variants using control and data flow analysis. In *MILITARY COMMUNICATIONS CONFERENCE, 2012-MILCOM 2012*, pages 1–6. IEEE.

Alazab, M., Layton, R., Venkataraman, S., and Watters, P. (2010). Malware detection based on structural and behavioural features of API calls.

Alazab, M., Venkatraman, S., Watters, P., and Alazab, M. (2011). Zero-day malware detection based on supervised learning algorithms of API call signatures. In *Proceedings of the Ninth Australasian Data Mining Conference-Volume 121*, pages 171–182. Australian Computer Society, Inc.

Altaher, A., Ramadass, S., and Ali, A. (2011). Computer virus detection using features ranking and machine learning. *Australian Journal of Basic and Applied Sciences*, 5(9):1482–1486.

Anderson, B., Quist, D., Neil, J., Storlie, C., and Lane, T. (2011). Graph-based malware detection using dynamic analysis. *Journal in Computer Virology*, 7(4):247–258.

Anderson, B., Storlie, C., and Lane, T. (2012). Improving malware classification: bridging the static/dynamic gap. In *Proceedings of the 5th ACM workshop on Security and Artificial Intelligence*, pages 3–14. ACM.

Annachhatre, C., Austin, T., and Stamp, M. (2014). Hidden Markov models for malware classification. *Journal of Computer Virology and Hacking Techniques*, pages 1–15.

Attaluri, S., McGhee, S., and Stamp, M. (2009). Profile hidden Markov models and metamorphic virus detection. *Journal in computer virology*, 5(2):151–169.

Austin, T. H., Filiol, E., Josse, S., and Stamp, M. (2013). Exploring hidden Markov models for virus analysis: A semantic approach. In *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, pages 5039–5048. IEEE.

Bascil, M. S. and Temurtas, F. (2011). A study on hepatitis disease diagnosis using multilayer neural network with Levenberg Marquardt training algorithm. *Journal of Medical Systems*, 35(3):433–436.

Bayer, U., Comparetti, P. M., Hlauschek, C., Kruegel, C., and Kirda, E. (2009). Scalable, behavior-based malware clustering. In *NDSS*, volume 9, pages 8–11. Citeseer.

Bayer, U., Kirda, E., and Kruegel, C. (2010). Improving the efficiency of dynamic malware analysis. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 1871–1878. ACM.

Bicego, M., Murino, V., and Figueiredo, M. A. (2004). Similarity-based classification of sequences using hidden Markov models. *Pattern Recognition*, 37(12):2281–2291.

Blanco, C., Lasheras, J., Valencia-García, R., Fernández-Medina, E., Toval, A., and Piattini, M. (2008). A systematic review and comparison of security ontologies. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 813–820. IEEE.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Cave, R. L. and Neuwirth, L. P. (1980). Hidden Markov models for English.

Chandramohan, M., Tan, H. B. K., and Shar, L. K. (2012). Scalable malware clustering through coarse-grained behavior modeling. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 27. ACM.

Christodorescu, M., Jha, S., Seshia, S. A., Song, D., and Bryant, R. E. (2005). Semantics-aware malware detection. In *Security and Privacy, 2005 IEEE Symposium on*, pages 32–46. IEEE.

Colbaugh, R., Glass, K., and Bauer, T. (2013). Dynamic information-theoretic measures for security informatics. In *Intelligence and Security Informatics (ISI), 2013 IEEE International Conference on*, pages 45–49.

Dahl, G. E., Stokes, J. W., Deng, L., and Yu, D. (2013). Large-scale malware classification using random projections and neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3422–3426. IEEE.

Damodaran, A., Di Troia, F., Visaggio, C. A., Austin, T. H., and Stamp, M. (2015). A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques*, pages 1–12.

Devesa, J., Santos, I., Cantero, X., Penya, Y. K., and Bringas, P. G. (2010). Automatic behaviour-based analysis and classification system for malware detection. In *ICEIS (2)*, pages 395–399.

Donner, M. (2003). Toward a security ontology. *IEEE Security & Privacy*, 1(3):6–7.

Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G. (1998). *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press.

Egele, M., Scholte, T., Kirda, E., and Kruegel, C. (2012). A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys (CSUR)*, 44(2):6.

Elhadi, A. A. E., Maarof, M. A., Barry, B. I., and Hamza, H. (2014). Enhancing the detection of metamorphic malware using call graphs. *Computers & Security*, 46:62–78.

Fenz, S. and Ekelhart, A. (2009). Formalizing information security knowledge. In *Proceedings of the 4th international Symposium on information, Computer, and Communications Security*, pages 183–194. ACM.

Garner, S. R. et al. (1995). WEKA: The waikato environment for knowledge analysis. In *Proceedings of the New Zealand computer science research students conference*, pages 57–64. Citeseer.

Grégio, A. R. A., de Geus, P. L., Kruegel, C., and Vigna, G. (2013). Tracking memory writes for malware classification and code reuse identification. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 134–143. Springer.

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. *International journal of human-computer studies*, 43(5):907–928.

Han, J., Kamber, M., and Pei, J. (2006). *Data mining: concepts and techniques.* Morgan kaufmann.

Hu, X., Chiueh, T.-c., and Shin, K. G. (2009). Large-scale malware indexing using function-call graphs. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 611–620. ACM.

Hu, X. and Shin, K. G. (2013). DUET: integration of dynamic and static analyses for malware clustering with cluster ensembles. In *Proceedings of the 29th Annual Computer Security Applications Conference*, pages 79–88. ACM.

Huang, H.-D., Acampora, G., Loia, V., Lee, C.-S., and Kao, H.-Y. (2011). Applying FML and fuzzy ontologies to malware behavioural analysis. In *Fuzzy Systems (FUZZ), 2011 IEEE International Conference on*, pages 2018–2025. IEEE.

Huang, H.-D., Chuang, T.-Y., Tsai, Y.-L., and Lee, C.-S. (2010). Ontology-based intelligent system for malware behavioral analysis. In *Fuzzy Systems (FUZZ), 2010 IEEE International Conference on*, pages 1–6. IEEE.

Huang, H.-D., Lee, C.-S., Wang, M.-H., and Kao, H.-Y. (2014). IT2FS-based ontology with soft-computing mechanism for malware behavior analysis. *Soft Computing*, 18(2):267–284.

Islam, R., Tian, R., Batten, L. M., and Versteeg, S. (2013). Classification of malware based on integrated static and dynamic features. *Journal of Network and Computer Applications*, 36(2):646–656.

Karampatziakis, N., Stokes, J. W., Thomas, A., and Marinescu, M. (2013). Using file relationships in malware classification. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 1–20. Springer.

Kephart, J. O., Sorkin, G. B., Arnold, W. C., Chess, D. M., Tesauro, G. J., White, S. R., and Watson, T. (1995). Biologically inspired defenses against computer viruses. In *IJCAI (1)*, pages 985–996.

Kim, K. and Moon, B.-R. (2010). Malware detection based on dependency graph using hybrid genetic algorithm. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1211–1218. ACM.

Kinable, J. and Kostakis, O. (2011). Malware classification based on call graph clustering. *Journal in computer virology*, 7(4):233–245.

Kolbitsch, C., Comparetti, P. M., Kruegel, C., Kirda, E., Zhou, X.-y., and Wang, X. (2009). Effective and efficient malware detection at the end host. In *USENIX Security Symposium*, pages 351–366.

Kolbitsch, C., Kirda, E., and Kruegel, C. (2011). The power of procrastination: detection and mitigation of execution-stalling malicious code. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 285–296. ACM.

Kolter, J. Z. and Maloof, M. A. (2004). Learning to detect malicious executables in the wild. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 470–478. ACM.

Krogh, A., Brown, M., Mian, I. S., Sjölander, K., and Haussler, D. (1994). Hidden Markov models in computational biology: Applications to protein modeling. *Journal of molecular biology*, 235(5):1501–1531.

Kuge, N., Yamamura, T., Shimoyama, O., and Liu, A. (2000). A driver behavior recognition method based on a driver model framework. Technical report, SAE Technical Paper.

Kumar, R. (2011). *Research Methodology-A step by step guide for beginners*. London: Sage, 3rd edition.

Langner, R. (2011). Stuxnet: Dissecting a cyberwarfare weapon. *Security & Privacy, IEEE*, 9(3):49–51.

Lee, W., Stolfo, S. J., and Chan, P. K. (1997). Learning patterns from unix process execution traces for intrusion detection. In *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 50–56.

Levinson, S. (1987). Continuous speech recognition by means of acoustic/phonetic classification obtained from a hidden Markov model. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'87.*, volume 12, pages 93–96. IEEE.

Liao, Y. and Vemuri, V. R. (2002). Using text categorization techniques for intrusion detection. In *USENIX Security Symposium*, volume 12, pages 51–59.

Liaw, A. and Wiener, M. (2002). Classification and regression by random forest. *R news*, 2(3):18–22.

Lin, C.-T., Wang, N.-J., Xiao, H., and Eckert, C. (2015). Feature selection and extraction for malware classification. *Journal of Information Science and Engineering*, 31:965–992.

Liu, A., Martin, C., Hetherington, T., and Matzner, S. (2005). A comparison of system call feature representations for insider threat detection. In *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC*, pages 340–347. IEEE.

Marian, T., Weatherspoon, H., Lee, K.-S., and Sagar, A. (2012). Fmeter: Extracting indexable low-level system signatures by counting kernel function calls. In *Middleware 2012*, pages 81–100. Springer.

McIntire, D. and Mundie, D. (2013). The MAL: A malware analysis lexicon.

Mehdi, S. B., Tanwani, A. K., and Farooq, M. (2009). IMAD: in-execution malware analysis and detection. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1553–1560. ACM.

Mitchell, M. (1998). *An introduction to genetic algorithms.* MIT press.

Moser, A., Kruegel, C., and Kirda, E. (2007). Limits of static analysis for malware detection. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 421–430. IEEE.

Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. (2011). Malware images: visualization and automatic classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, page 4. ACM.

Park, Y., Reeves, D., Mulukutla, V., and Sundaravel, B. (2010). Fast malware classification by automated behavioral graph matching. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, page 45. ACM.

Park, Y., Reeves, D. S., and Stamp, M. (2013). Deriving common malware behavior through graph clustering. *Computers & Security*, 39:419–430.

Qiao, Y., Yang, Y., He, J., Tang, C., and Liu, Z. (2014). CBM: Free, automatic malware analysis framework using api call sequences. In *Knowledge Engineering and Management*, pages 225–236. Springer.

Rabiner, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Rabiner, L. and Juang, B.-H. (1986). An introduction to hidden Markov models. *ASSP Magazine, IEEE*, 3(1):4–16.

Ranveer, S. and Hiray, S. (2015). Comparative analysis of feature extraction methods of malware detection. *International Journal of Computer Applications*, 120(5).

Raskin, V., Hempelmann, C. F., Triezenberg, K. E., and Nirenburg, S. (2001). Ontology in information security: a useful theoretical foundation and methodological tool. In *Proceedings of the 2001 workshop on New security paradigms*, pages 53–59. ACM.

Ravi, S., Balakrishnan, N., and Venkatesh, B. (2013). Behavior-based malware analysis using profile hidden Markov models. In *Security and Cryptography (SECRYPT), 2013 International Conference on*, pages 1–12.

Rieck, K., Holz, T., Willems, C., Düssel, P., and Laskov, P. (2008). Learning and classification of malware behavior. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 108–125. Springer.

Rieck, K., Trinius, P., Willems, C., and Holz, T. (2011). Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4):639–668.

Rojas, R. (1996). *Neutral Networks: A Systematic Introduction.* Springer.

Santos, I., Brezo, F., Ugarte-Pedrero, X., and Bringas, P. G. (2013). Opcode sequences as representation of executables for data-mining-based unknown malware detection. *Information Sciences*, 231:64–82.

Santos, I., Penya, Y. K., Devesa, J., and Bringas, P. G. (2009). N-grams-based file signatures for malware detection. In *ICEIS (2)*, pages 317–320.

Saxe, J., Mentis, D., and Greamo, C. (2012). Visualization of shared system call sequence relationships in large malware corpora. In *Proceedings of the Ninth International Symposium on Visualization for Cyber Security*, pages 33–40. ACM.

Schultz, M. G., Eskin, E., Zadok, E., and Stolfo, S. J. (2001). Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 38–49. IEEE.

Shafiq, M. Z., Khayam, S. A., and Farooq, M. (2008). Embedded malware detection using Markov n-grams. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 88–107. Springer.

Shankarapani, M. K., Ramamoorthy, S., Movva, R. S., and Mukkamala, S. (2011). Malware detection using assembly and API call sequences. *Journal in computer virology*, 7(2):107–119.

Stopel, D., Boger, Z., Moskovitch, R., Shahar, Y., and Elovici, Y. (2006). Application of artificial neural networks techniques to computer worm detection. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 2362–2369. IEEE.

Storlie, C., Anderson, B., Vander Wiel, S., Quist, D., Hash, C., Brown, N., et al. (2014). Stochastic identification of malware with dynamic traces. *The Annals of Applied Statistics*, 8(1):1–18.

Tamersoy, A., Roundy, K., and Chau, D. H. (2014). Guilt by association: large scale malware detection by mining file-relation graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1524–1533. ACM.

Tian, R., Islam, R., Batten, L., and Versteeg, S. (2010). Differentiating malware from cleanware using behavioural analysis. In *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*, pages 23–30. IEEE.

Trinius, P., Holz, T., Gobel, J., and Freiling, F. C. (2009). Visual analysis of malware behavior using treemaps and thread graphs. In *Visualization for Cyber Security, 2009. VizSec 2009. 6th International Workshop on*, pages 33–38. IEEE.

Undercoffer, J., Joshi, A., and Pinkston, J. (2003). Modeling computer attacks: An ontology for intrusion detection. In *Recent Advances in Intrusion Detection*, pages 113–135. Springer.

Wang, H.-T., Mao, C.-H., Wei, T.-E., and Lee, H.-M. (2013). Clustering of similar malware behavior via structural host-sequence comparison. In *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual*, pages 349–358. IEEE.

Warrender, C., Forrest, S., and Pearlmutter, B. (1999). Detecting intrusions using system calls: Alternative data models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 133–145. IEEE.

Willems, C., Holz, T., and Freiling, F. (2007). Toward automated dynamic malware analysis using CWSandbox. *IEEE Security and Privacy*, 5(2):32–39.

Wong, W. and Stamp, M. (2006). Hunting for metamorphic engines. *Journal in Computer Virology*, 2(3):211–229.

Xing, Z., Pei, J., and Keogh, E. (2010). A brief survey on sequence classification. *ACM SIGKDD Explorations Newsletter*, 12(1):40–48.

Ye, Y., Li, T., Chen, Y., and Jiang, Q. (2010). Automatic malware categorization using cluster ensemble. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 95–104. ACM.

Yoo, I. (2004). Visualizing windows executable viruses using self-organizing maps. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 82–89. ACM.

You, I. and Yim, K. (2010). Malware obfuscation techniques: A brief survey. In *BWCCA*, pages 297–300.